RIVISTA +SUPPLEMENTO +2 CD-ROM EURO 7,70

I CORSI DA ZERO PER IMPARARE: VISUAL BASIC, C++, C#, VB.NET, JAVA E JSP

RIVISTA+SUPPLEMENTO "PROGRAMMARE IL WEB"+2CD € 7,70

Periodicità mensile • FEBBRAIO 2003 • ANNO VIII, N.2 (66)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioProgrammo

Grafica Digitale e Visual Basic

Sviluppiamo una completa applicazione di fotoritocco

ALL'INTERNO IL SUPPLEMENTO

XML & FLASH

Impariamo a integrare le tecnologie che hanno rivoluzionato il Web

WinZIP fai da te!

Le tecniche di compressione utilizzate dalle librerie Open Source

SISTEMA

- Impronte digitali: realizziamo un'applicazione VB per il riconoscimento e la catalogazione
- Tecniche per ottimizzare le applicazioni Delphi

ELETTRONICA

- Lego Mindstorms: un laboratorio di robotica sulla tua scrivania
- Moduli Rabbit: elettrodomestici, luci, allarmi, telecamere, ora li controlli dal Web!
- Realizziamo un software per il controllo dei motori passo passo

ADVANCED

Reti locali, cablate, wireless: analizziamone il comportamento con Network Simulator

HACKER

Scovare le password di alcuni router ADSL

ISSN 1128-594X

è una pubblicazione EDIZIONI MASTER

CONCORSO CROBOTS: I RISULTATI DEL CONCORSO 2002

Contents

Anno VII - n. 2 (66) Febbraio 2003

Povere le mie orecchie!?

Sovente le mie orecchie hanno dovuto ascoltare frasi campate in aria, elucubrazioni mentali e disquisizioni su nuovi fantomatici dispositivi che cambieranno il modo di vivere: TV virtuali che si materializzano al solo pensiero, ascensori spaziali in grado di catapultarci sulla luna, auto volanti e chi più ne ha, più ne metta. Questo nuovo anno porta con se nuovi "pettegolezzi": orologi da polso e magneti da attaccare al frigorifero, sistemi in grado di integrare le comuni operazioni quotidiane, e l'individuo stesso, all'interno della grande Rete.

Fantascienza o realtà? Sara forse questa, una delle tante altre novelle che le mie povere orecchie dovranno ancora una volta sopportare? I dubbi sono tanti ma, forse questa volta, ci sarà da scommettere qualcosa, soprattutto se a prospettare il progetto è lui, l'innominabile: Bill Gates: *"Da qui a breve ogni dispositivo, piccolo o grande che esso sia, sarà dotato del nuovo .NET Compact Framework, tutto e tutti saranno in grado di interfacciarsi alla Rete"*. Frasi che non restano nella fervida immaginazione del buon zio Bill ma che trovano raffronto al Consumer Electronics Show, dove, per la prima volta, sono stati presentati i primi dispositivi che utilizzeranno la nuova tecnologia. Fossil, Citizen e Suunto, produttori d'orologi da polso, prevedono di mettere in commercio modelli predisposti ad accogliere SPOT (questo il nome del progetto) entro la fine dell'anno. Gli orologi si collegheranno al PC per compiere una serie d'operazioni: regolazione automatica dell'ora, download di software e collegamento wireless a dati streaming trasmessi mediante segnali radio FM, per ricevere informazioni di qualunque genere. Per non parlare delle calamite per il frigorifero, veri gioiellini che ricevono informazioni wireless sul traffico locale o sulle specialità dei ristoranti della zona. Scommettiamo che a breve sarà possibile, semplicemente parlando al proprio orologio, scongelare la cena e azionare il forno a micronde? Scommetto le mie orecchie!

Gianfranco Forlino (gforlino@edmaster.it)

ioProgrammo

Anno VII - N.ro 2 (66) - Febbraio 2003 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 - Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo

Dir. Editoriale Massimo Sesti
Dir. Responsabile Romina Sesti
Product Manager Antonio Meduri
Editor Gianfranco Forlino
Coordinatore Redazionale Raffaele del Monaco
Redazione Thomas Zaffino, Antonio Pasqua
Collaboratori S. Ascheri, M. Autiero, R. Bandiera, L. Buono, P. Canini, E. Cobisi, F. Grimaldi, M. Del Gobbo, E. Florio, A. Marroccelli, S. Meschini, G. Palumbo, A. Panella, A. Pelleriti, C. Pelliccia, P. Perrotta, S. Serra, L. Spuntoni, G. Uboldi, F. Vaccaro.
Per l'inserto hanno collaborato M. Battista, A. Cangiano, P. Capitani, C. Giustozzi, F. Mestrone, G. Uboldi
Segreteria di Redazione Veronica Longo

REALIZZAZIONE GRAFICA CROMATIKA Srl
C.da Lecco, zona ind. - 87030 Rende (CS)
Tel. 0984 8319 - Fax 0984 8319225
Coord. grafico: Paolo Cristiano
Coord. tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

PUBBLICITÀ Edizioni Master S.r.l.
Responsabile Vendite Ernesto Redaelli
Agenti Vendita Elisabetta Febbruo, Serenella Scarpa, Cornelio Morari
Segreteria Ufficio Vendite Daisy Zonato
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321612 - Fax 02 8321764
e-mail: advertising@edmaster.it

EDITORE Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321482 - Fax 02 8321699
Sede di Cosenza: C.da Lecco, zona ind. - 87030 Rende (CS)
Amministratore Unico: Massimo Sesti
Responsabile Amministraz. e Finanza: Benedetto Celsa
Produzione e Logistica: Michele Carere
Diffusione: Alessandra Cervello
Marketing: Giuseppina Bruno, Leonardo Petrone, Antonio Meduri

ABBONAMENTO E ARRETRATI
Costo abbonamento annuale (11 numeri): Italia € 84,70 Promozione Sconto 30% € 59,00
Costo abbonamento annuale (11 numeri): estero € 169,40
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5,16 spese (spedizione con corriere). Verificare la disponibilità delle copie arretrate allo 028321482. La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 028321699, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/ EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a: Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel.02 8321482
e-mail: servizioabbonati@edmaster.it

Stampa: Elcograf Industria Grafica (LC)
Stampa CD-Rom: Disctronics Italia (MI)
Distribuzione per l'Italia: Parrini & C S.p.A. - Roma

Finito di stampare nel mese di Gennaio 2003

L'Universo Tecnologico
www.itportal.it

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, Quale Computer, DVD Magazine, Office Magazine, ioProgrammo, Linux Magazine, Softline Software World, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, Il CD-Rom di Idea Web, I Corsi di Win Magazine, Le Collection.

News

Bill Gates presenta OneNote, la nuova applicazione per prendere appunti in modo più produttivo

OneNote unisce l'efficienza tipica dei contenuti digitali alla possibilità di prendere appunti in modo flessibile

In occasione del COMDEX 2002, Bill Gates, Chairman and Chief Software Architect di Microsoft Corp., ha annunciato la prossima disponibilità di Microsoft One Note, un'applicazione che permette agli utenti di prendere appunti, organizzarli e utilizzarli in modo più produttivo. OneNote, sul mercato a metà del 2003, consentirà di prendere appunti su un desktop o su un computer portatile, mentre gli utenti di Tablet PC potranno acquisire anche note scritte a mano, immagini e diagrammi. "OneNote combina la flessibilità di un taccuino cartaceo con l'efficienza tipica del contenuto digitale, mantenendo lo stile con cui ciascuno di noi fissa sulla carta i propri pensieri e organizza gli appunti", ha dichiarato Jeff Raikes, Group Vice President of Productivity and Business Services di Microsoft. "Con la creazione di nuove applicazioni come OneNote, Microsoft è impegnata nel mantenere la suite Office sempre aggiornata e al passo con i tempi, e nel migliorare la produttività degli information worker." Da un recente studio condotto da Microsoft Research, risulta che il 91% degli information worker – tutti coloro che nello svolgimento della propria attività professionale gestiscono l'informazione attraverso il PC e i dispositivi digitali - intervistati prende regolarmente appunti, il 26% li trasferisce in messaggi di posta elettronica, mentre il 23% ha ammesso di non riuscire sempre a ritrovare le informazioni desiderate. Inoltre, il 36% ha dichiarato di essere pronto ad adottare un nuovo sistema di appunti.

VeriSign annuncia un toolkit per la sicurezza nei WebServices

Una implementazione Open Source di un interessante tool che consentirà agli sviluppatori di gestire problematiche come firma digitale e cifratura

VeriSign ha introdotto una implementazione completamente gratuita e Open Source di un toolkit rivolto agli sviluppatori che semplifica l'integrazione della firma digitale e della cifratura all'interno di Web Services. VeriSign afferma che l'implementazione sarà distribuita attraverso delle specifiche librerie open source che daranno agli sviluppatori la possibilità di costruire Web Service "sicuri" che utilizzino lo standard WS-Security. Il *Trust Service Integration Kit* è un tool Java-based che permette di gestire XML Signature, XML Encryption ed altre funzioni relative alla sicurezza. Il TSIK è composto da tre componenti fondamentali: il *Messaging Framework*, il *Trust Layer* ed alcune risorse specifiche per XML.

- Il *Messaging Framework* può essere utilizzato per specificare firma e chiave di cifratura per garantire autenticazione, integrità dei dati e riservatezza delle comunicazioni.
- Il *Trust Laywe* fornisce delle API per la sicurezza dei messaggi XML attraverso una infrastruttura a chiave pubblica (PKI), e include le implementazione delle specifiche W3C per la firma digitale e la cifratura XML.
- Le risorse per XML permettono di manipolare direttamente file XML, navigare nella struttura dei documenti, validare gli schema oltre a semplificare l'interfacciamento con i parser.

www.verisign.com

IBM pronta a pubblicare il codice della nuova tecnologia di Storaging

Con lo scopo di attirare nuovi sviluppatori, IBM sta approntando una versione Open Source di Storage Tank

I ricercatori di IBM sono al lavoro per proporre una versione Open Source di una delle più attese release del 2003: Storage Tank. Una tecnologia sviluppata per meglio gestire i sistemi di storaging esistenti e facilitare la cooperazione fra piattaforma diverse. Il modello di upgrade proposto da IBM si può definire dunque rivoluzionario: Storage Tank consente di integrare vecchi e nuovi sistemi, il tutto a vantaggio dei costi e della praticità. Storage Tank è una sorta di file system virtuale che si basa sul concetto di metadata, attraverso cui gestisce posizione, dimensione e permessi associati ai file immagazzinati. La caratteristica più interessante che Storage Tank offre consiste nella possibilità di far interagire sistemi assolutamente eterogenei: gli stessi file saranno accessibili da sistemi operativi differenti e potranno essere salvati su Windows, AIX e Linux in modo del tutto trasparente agli utenti. L'eterogeneità non si ferma ai server e ai sistemi operativi, ma arriva al supporto per dischi e nastri (fondamentale per preservare i sistemi già esistenti) di qualsiasi marca e modello. Storage Tank sarà dunque capace di scendere a livello dei singoli blocchi che compongono un file, identificarne il formato e tradurlo in modo da renderlo compatibile al dispositivo che lo aveva richiesto.

www.ibm.com/it/

Il PocketPC ci fa le multe

Il Palmare sta cambiando in meglio il nostro mondo

L'applicazione PocketPM permette agli agenti di Polizia Municipale di sostituire il vecchio blocchetto delle multe con un innovativo palmare, migliorando l'efficienza del loro lavoro e fornendo un migliore servizio al cittadino.

Il prodotto software, realizzato da una giovane e innovativa società di Cosenza, la NoTangle, informatizza l'intero processo di attività degli agenti di Polizia Municipale, dal rilevamento informatico delle infrazione al Codice della Stradare alla gestione informatizzata del Back Office con un notevole risparmio di tempo e danaro. L'applicazione client, sviluppata per PocketPC utilizzando l'eMbedded Visual C++ e SQLCE 2.0, consente tramite una interfaccia amichevole di registrare i dati dell'infrazione e di scattare una foto attraverso una camera digitale integrata con il dispositivo. I dati raccolti sono successivamente sincronizzati con il server centrale utilizzando una connessione sicura GPRS ed elaborati in modo efficace e veloce. L'applicazione lato server è stata interamente sviluppata sfruttando le nuove opportunità offerte dalla piattaforma .NET di Microsoft.

FileMaker presenta la gamma educational

Al MacWorld EXPO di San Francisco, presentata la gamma di soluzioni database K12 per le scuole

Grazie alle numerose e potenti caratteristiche, ai template per l'insegnamento pronti per l'uso e alle centinaia di soluzioni per la vasta comunità di esperti sviluppatori, FileMaker Pro 6 rende ancora più economica ed efficiente la gestione e la condivisione delle informazioni, offrendo soluzioni concrete a migliaia di scuole nei principali distretti del Nord America.

Il mese scorso FileMaker ha lanciato un nuovo sito web, K-12 Education, che offre soluzioni software gratuite per l'insegnamento, ultime notizie che riguardano le proposte di FileMaker per il mondo delle scuole, aggiornamenti sulle conferenze e sugli eventi dove gli insegnanti possono assistere a dimostrazione sull'uso di FileMaker e offerte speciali dei prodotti FileMaker destinate agli insegnanti e amministratori nel segmento K-12.

www.filemaker.com/education

Macromedia Director diventa MX

Disponibile la versione MX di Macromedia Director, il più noto software di authoring multimediale

La nuova versione di Macromedia Director è stata arricchita di funzionalità per gli utenti che realizzano prodotti interattivi su CD-ROM destinati alla pubblicazione su Internet. Director MX si integra perfettamente con tutti gli altri prodotti della gamma MX. Il software permette la creazione di contenuti che possano essere distribuiti dovunque, sia online che offline. Supporta tutti formati audio, video, 3D e grafici più diffusi e permette agli sviluppatori di utilizzare stream video senza limiti di durata, garantendo il supporto QuickTime, RealVideo e AVI. Gli sviluppatori possono lanciare Flash direttamente da Director MX e quindi di effettuare rapidi cambiamenti dei contenuti o controllare Flash attraverso l'utilizzo di Lingo. Inoltre, il software si integra con le tecnologie server di Flash, e in particolare con Flash Remoting MX, permettendo di ottenere una connessione ad elevate prestazioni tra ColdFusion MX e il Player Shockwave e con Flash Communication Server MX per la gestione di giochi multi utente, streaming video Flash e collaborazioni in tempo reale.

www.macromedia.com/it/software/director/

WebSphere 5 supporta Soap

Uscita la quinta release dell'application server di IBM, con funzioni di autodiagnosi e autoriparazione e un miglior supporto ai web service

Rilasciato WebSphere Application Server 5.0, con funzioni per "lautonomic computing", un supporto potenziato per i web service. Secondo IBM, WebSphere 5.0 costituirà la piattaforma di base per il software "On Demand" di IBM e sarà integrato con il database Db2, le applicazioni Tivoli per la gestione della rete e il software Lotus per la collaborazione, puntando a definire nuovi standard in termini di prestazioni, scalabilità, affidabilità, facilità di manutenzione ed interfaccia d'uso. Inoltre gli strumenti di sviluppo, strettamente integrati nell'ambiente, permettono agli sviluppatori di essere più veloci ed efficaci nella creazione di applicazioni J2ee". Dal punto di vista commerciale, è stata aggiunta una nuova linea di prodotti, la cosiddetta *Express* che permette a IBM di estendere la propria offerta agli Isv e alle aziende che hanno bisogno di un ambiente J2ee, ma anche di un paradigma di sviluppo per le applicazioni semplificato. I nuovi tool, infatti, rendono più semplice lavorare sui web service, "automatizzando" il processo di sviluppo, fino a limitare al minimo la scrittura di codice. In più, i wizard di WebSphere Studio danno l'opportunità di ispezionare il codice prima di eseguirlo. WebSphere 5 è anche una buona piattaforma di integrazione perché ha il pieno supporto per J2ee Connection Architecture 1.0. IBM ha aggiunto il supporto per Web Service Invocation Framework, una tecnologia che permette di richiamare e sviluppare servizi Web su varie reti e protocolli di trasporto, e Axis 3.0, un nuovo sistema per i web service ad alta velocità in grado di gestire richieste Soap, il più importante protocollo di trasporto per questi servizi. WebSphere 5.0 include anche Ibm Web Services Gateway per amministrare e rendere sicuri i Web service; un repository privato Uddi *(Universal Description, Discovery and Integration)*; una tecnologia per il workflow dei Web service basata su Bpel4Ws *(Business Process Execution Language)* for Web Services.

www.ibm.com/it/

Batosta di Sun a Microsoft

Per ordine di un giudice, la Java Virtual Machine di Sun dovrà essere inclusa nel sistema operativo di Windows e nel browser Internet Explorer

La corte di giustizia del Maryland ha dato ragione alla Sun in una causa intentata contro la Microsoft per violazioni della legge antitrust e del copyright. L'oggetto della causa era l'implementazione di una versione proprietaria del linguaggio Java. Ora Microsoft dovrà distribuire Java

"genuino". Il contenzioso tra le due aziende risale sin dal Marzo 1996, quando Microsoft strinse un accordo con Sun per distribuire prodotti basati sulla tecnologia Java, allora molto in voga. La natura trasversale della piattaforma Java, che non dipende dal sistema operativo, venne giudicata da Microsoft come un serio rischio alla posizione di predominio di Windows. Microsoft tentò quindi di togliere a Sun il controllo di Java, trasformando Java in un modo per scrivere applicazioni per Windows. Ma Sun replicò con una clausola che obbligava i prodotti Java distribuiti da terzi a superare un "core test" di Sun. Microsoft apportò quindi delle modifiche alla Virtual Machine, creandosi una propria versione (MSJVM). La piattaforma Java originale continuò ad essere distribuita soprattutto grazie a Netscape Navigator. Sun iniziò un'azione legale che portò ad un primo successo il 24 Marzo 1998, corte distrettuale della California del Nord. Nella sentenza, si proibiva a Microsoft di utilizzare la dizione "compatibile Java" nella distribuzione di MSJVM. La corte d'appello annullò però la sentenza, pur senza negare l'incompatibilità di MSJVM con Java. Sun si rifiutò di fornire il sorgente aggiornato a Microsoft. MSJVM, distribuito da Microsoft, divenne presto obsoleto rispetto agli sviluppi di Java. MSJVM è più lento, più soggetto a "crash" e con meno funzioni rispetto alle versioni più recenti di Java. Il 23 Gennaio 2001 Microsoft e Sun trovarono un accordo che permise di mettere fine alla controversia legale in California, ma Microsoft decise di lavorare ad un'alternativa a Java, quello che oggi noi conosciamo come .NET framework. Anche in questo caso si tratta di una piattaforma "trasversale" e che utilizza un linguaggio diverso da Java. Il 13 Febbraio 2002 Microsoft cominciò la distribuzione commerciale del primo prodotto contenente il framework .NET, ossia Visual Studio .NET. Nello stesso tempo Microsoft smise di fornire MSJVM, che infatti non è incluso in WindowsXP, anche se è poi stato reinserito nel Service Pack 1, con annuncio che comunque nel prossimo futuro sarà abbandonato. Sun reagì affermando che Microsoft stava sfruttando i vantaggi derivanti dalla precedente attività illegale (rispetto alla legge antitrust) con la quale aveva frammentato la piattaforma Java e distrutto i canali di distribuzione di cui Sun si era avvalsa. Sun intraprese quindi un'altra azione legale, richiedendo che Microsoft distribuisse l'ultima versione, fornitale a costo zero, del binario (il sorgente non essendo più fornito a Microsoft già da un po' di tempo) di Java (Java Runtime Environement: JRE) con tutti i pacchetti Windows e con Internet Explorer; che JRE venisse installato ed abilitato automaticamente; che Microsoft rendesse disponibile JRE tramite il sito Windows Update; e che Microsoft annunciasse la prossima disponibilità di nuove versioni di Windows e di Internet Explorer con almeno 120 giorni di anticipo, e non modificasse in alcun modo il codice binario di JRE. La corte di giustizia del Maryland ha dato ragione a Sun e quindi, se la decisione non sarà annullata in appello (Microsoft ha già annunciato che presenterà ricorso) gli utenti Windows dovrebbero presto avere a disposizione JRE tramite il sito di aggiornamento di Windows. Non solo: i prossimi CD di Windows, nonché le distribuzioni di Internet Explorer, conterranno JRE.

http://it.sun.com

Intel presenta i nuovi compilatori

Disponibile la versione 7.0 dei compilatori Intel C++ e Intel Fortran per Windows e Linux

I nuovi compilatori Intel permettono di sviluppare applicazioni che possano avvalersi della tecnologia Hyper-Threading di Intel, grazie alla quale si possono eseguire più attività contemporaneamente. I compilatori sono stati ottimizzati per i processori Intel Itanium 2, Intel Pentium 4 e Intel Xeon, e consentono di realizzare applicazioni compilate in modo più efficace per velocizzare le operazioni. Tra le varie applicazioni che potranno beneficiare dell'ottimizzazione, i

programmi commerciali orientati alle transazioni, le applicazioni finanziarie ad elaborazione intensiva e quelle tecniche e scientifiche, i programmi multimediali digitali, i videogame e gli effetti speciali. I nuovi compilatori supportano funzioni di Compaq Visual Fortran, tra cui la compatibilità a livello di riga di comando, e prevedono l'integrazione con Microsoft Visual Studio. La versione di Linux prevede la compatibilità a livello di GNU con C++ con l'adozione dell'interfaccia binaria delle applicazioni C++. I nuovi compilatori Intel versione 7.0 comprendono inoltre un'opzione di parallelizzazione automatica, tramite la quale, nelle applicazioni, viene automaticamente ricercata la possibilità di creare più thread esecutivi e miglioramenti per il supporto OpenMP, standard di settore che consente l'uso di direttive di alto livello per semplificare la creazione e la gestione di software multithreaded.
I compilatori Intel C++ versione 7.0 per Windows e Linux sono già disponibili al prezzo di listino consigliato di 399 dollari USA ciascuno. I compilatori Intel Fortran versione 7.0 per Windows e Linux sono altresì disponibili al prezzo di listino consigliato di 499 e 699 dollari rispettivamente. I compilatori possono essere acquistati tramite download presso Intel e i rivenditori di tutto il mondo, e saranno presto disponibili anche su CD-ROM.

www.intel.com/software/products/

Allarme BSA: giovani pirati in aumento

La pirateria informatica tra i ragazzi europei allarma la Business Software Alliance

La Business Software Alliance ha espresso forte preoccupazione per il numero crescente di casi di pirateria software su Internet in Europa che vedono coinvolti i minori. Tre casi sottoposti recentemente dalle polizie locali all'attenzione dell'European Internet Investigation Team della BSA, il gruppo di investigatori dedicato alla ricerca di siti internet illegali, hanno visto infatti la partecipazione di minori ad attività commerciali illegali. In tutte e tre le occasioni le autorità hanno intrapreso opportune azioni, mentre altri casi simili sono attualmente al vaglio degli inquirenti. Nel Regno Unito uno studente sedicenne è stato incriminato per aver venduto software pirata su siti d'aste online, mentre in Svizzera un minore è stato arrestato per aver pubblicizzato e venduto software pirata su un sito Web specializzato in piccoli annunci. Sempre in questo paese, uno studente minorenne è stato incriminato perché vendeva CD contenenti software "pirata". In tutti i casi la magistratura ha imposto severe pene pecuniarie alle quali, in una circostanza, si sono aggiunte anche restrizioni sull'uso futuro di Internet. Studi recenti dimostrano che alcuni genitori giudicano innocuo questo tipo di attività e tendono a chiudere un occhio con il risultato che i minori vengono incriminati e i genitori sono costretti a pagare multe elevate. Organizzazioni come la BSA compiono ogni sforzo per spiegare ai giovani i rischi insiti in queste attività, ma senza l'attiva collaborazione dei genitori si tratta spesso di sforzi vani.

Il Tablet PC diventa un bloc-notes evoluto

Alias Wavefront annuncia uno strumento software intuitivo per trasformare il Tablet PC in un bloc notes digitale

Alias SketchBook Pro permetterà agli utenti di cambiare i pennelli elettronicamente, di fare e rifare velocemente i propri lavori, di manipolare immagini sovrapposte e di integrare gli schizzi digitali dall'interno del mondo Windows XP. Pensato appositamente per i Tablet PC, Alias SketchBook Pro offre la possibilità di fare bozzetti, annotare informazioni e immagini, presentare il proprio lavoro ogni volta che si realizza una nuova idea creativa: Alias SketchBook Studiato per una vasta gamma di utenti aziendali e grafici professionali, Alias Sketchbook Pro si basa sulla tecnologia grafica sviluppata da Alias | Wavefront per i software Maya e Studio Tools. Supportato da un'interfaccia utente basata sui gesti che offre agli utenti la possibilità di cambiare strumenti e colori, utilizzare la funzione di zoom, salvare e modificare gli schizzi, Alias Sketchbook Pro introduce un insieme di strumenti di sketching semplice ma completo.
È possibile creare una raccolta di schizzi che possono essere visualizzati, organizzati e distribuiti facilmente, o effettuare presentazioni formali e improvvisate di sketch e immagini e registrare qualsiasi feedback istantaneamente. Infine, Marking Menu è un'interfaccia utente ergonomica che riconosce i movimenti naturali della mano per selezionare i comandi. Alias Sketchbook Pro sarà disponibile nei primi mesi del 2003.

www.aliaswavefront.com/sketchbook
www.microsoft.com/tabletpc

Hi Tech: più imprese che esperti!

Sono ancora numerose le figure con competenza adeguata che mancano nel settore Ict italiano...

Il rapporto 2002 Occupazione e formazione nell'Ict nasce con l'auspicio di analizzare dinamiche, caratteristiche e opportunità del mondo occupazionale dell'information & communication technology italiano. La capacità di sviluppo delle piccole e medie imprese e il dinamismo del settore richiedono uno sforzo di riqualificazione sia delle aziende stesse sia del personale coinvolto, come ha spiegato Giulio Koch, presidente di Assinform. È necessario avviare un processo di rinnovamento che coinvolga tutti i protagonisti per adeguare la formazione di base.
Secondo le elaborazioni di Unioncamere i lavoratori delle imprese Ict quest'anno sono 598.000 (di cui ben il 52,8% è impiegato nelle aziende di software e servizi), dato che porta il settore al settimo posto nella classifica dell'industria privata per numero di addetti. Le imprese che hanno sistemi informativi arretrati investono maggiormente in corsi che hanno come meta l'alfabetizzazione, le altre investono di più in formazione tecnica e professionale, ha commentato Capitani. Dal rapporto emerge un dinamismo forte sulla crescita della formazione, ma c'è un livello di inadeguatezza decrescente, perché si registra un processo di rapida obsolescenza delle competenze. Inoltre la compenetrazione tra le tecnologie e i processi ne velocizza ancora di più l'obsolescenza.
Secondo il rapporto, la carenza di risorse specializzate genera diversi problemi: primo fra tutti l'impossibilità di realizzare un progetto. Ma ci sono anche la perdita di competitività verso i concorrenti, quella d'immagine verso i clienti per il ritardo del progetto, o la possibilità di prendere decisioni sbagliate.
Nella formazione non c'è sistematicità, afferma Franco Patini, presidente An@sin. Si passa da esempi di eccellenza ad arretratezza. Manca una diffusione omogenea e qualitativamente ideale. Inoltre, il Paese non investe nell'istruzione, perché manca la capacità di lavorare insieme. Senza dimenticare che gli investimenti hanno valori marginali e che siamo in presenza di una scarsa coscienza del valore strategico della formazione.
Non bisogna cercare il ritorno di valore della conoscenza, bensì il valore della competenza.

www.assinform.it
www.unioncamere.it

Frasi anagrammate

Riveste particolare interesse sia nel campo enigmistico che in quello della programmazione, la produzione automatica di anagrammi multiparola, ovvero che in generale trasformano frasi in frasi.

"Il mago di fibra": metafora postmoderna di esperto di reti ad alta velocità, "Maglio di fibra": una variante più aggressiva della figura precedente, "Figlia morbida": delicata immagine familiare, "Fa orli di gambi": attività artigiana che ricorda ambienti bucolici e ancora "Il dio fra gambi" soggetto a limite tra il floreale e il blasfemo; ecco un insieme di figure a me intrinsecamente legate, diciamo pure mie strette parenti, frasi che con me, "Fabio Grimaldi", sono in forte relazione. Si tratta, infatti, di anagrammi multiparola del mio nome e cognome o se preferite frasi anagrammate. Gli amanti dell'enigmistica sanno benissimo che stiamo parlando di un intrigante gioco che si fa con parole. Qualche altro esempio? Presto serviti. Un anagramma di George Bush è "He bugs Gore", che se valutato prima delle elezioni presidenziali assume un perverso carattere premonitore. Con la presenza della double (w), si innesca l'interessante anagramma "He grew bogus", che svela la presunta personalità del president, sarà davvero cresciuto falso? chissà. Stiamo scherzando ovviamente, tutti giochi di parole, come l'anagramma di "Madonna Louise Ciccone" la pop star di origine italiana che dopo una rimescolata di lettere si trasforma in "Occasional nude income". Mentre, curioso è l'anagramma di un conosciuto sistema operativo di mamma Microsoft, infatti, "Windows Me" diventa nuova saggezza, ossia "new wisdom". In questo appuntamento tenteremo un salto di qualità, oltre a produrre un semplice anagramma, dove per semplice intendo un anagramma che è una nuova permutazione delle lettere di una singola parola che a sua volta ha un senso compito, tenteremo di sviluppare un algoritmo per la generazione di frasi anagrammate. Come sarà risultato chiaro dai precedenti esempi, un anagramma multiparola o una frase anagrammata è un particolare anagramma per il quale la stringa di origine e quella di destinazione non sono singole parole, come nel caso standard, ma più in generale insiemi di parole. Vedremo che le tecniche algoritmiche per la produzione di questi interessanti tipi di anagrammi si basano su quelle sviluppate per il caso base, studiate nel numero scorso e di cui daremo un breve ma significativo cenno nel prossimo paragrafo.

LA VERSIONE ZIPPATA DELLA PUNTATA PRECEDENTE

Per comprendere le tecniche che svilupperemo, per chi non ha letto il precedente numero, è necessario conoscere alcune fondamentali nozioni. In tal modo il presente articolo sarà completamente indipendente dal precedente anche se di questo ultimo si consiglia quanto meno una rapida lettura. La ricerca di anagrammi assume una complessità non esponenziale con una semplice quanto efficace idea. La stringa da anagrammare anziché essere confrontata in tutte le sue permutazioni con tutte le parole presenti in un dizionario, che sarebbe una operazione di elevatissima complessità per la natura esponenziale del calcolo combinatorio, viene confrontata con una firma associata alla parola. La firma di una parola non è altro che la permutazione di lettere della stessa in modo ordinato, ad esempio la firma della parola *anagramma* è *aaaagmmnr.* Una volta individuata per ogni parola la sua firma, basta accorpare (collassare) le firme uguali mantenendo la lista per ognuna di esse delle parole che le hanno generate. Infine, bisogna ordinare la struttura per firma. Abbiamo ottenuto quello che è stato definito dizionario delle firme. Qualora si vuole ricercare un anagramma basta prendere la parola da anagrammare (sorgente) calcolarne la firma ed effettuare una ricerca binaria sul dizionario delle firme, quindi esaminare se esiste la liste delle parole associate a quella firma. Insomma, fatto salva la produzione di un dizionario delle firme che è una operazione da effettuare una tantum e che comunque ha una complessità computazionale polinomiale, la ricerca avrà complessità ancora minore che come si sa, per la ricerca binaria è di tipo logaritmico. Nell'articolo precedente si potranno trovare utili approfondimenti nonché la struttura dell'algoritmo risolutore che sicuramente chiarirà eventuali dubbi.

PRODURRE ANAGRAMMI MULTIPAROLA

Anagrammare una frase producendo un'altra frase non è altro che una naturale estensione del metodo appena descritto. Sarà necessario ovviamente prestare alcune attenzioni. Anche questa volta invito chi volesse approfondire l'argomento a visitare il sito allestito da uno dei pionieri dell'anagrammatica in Italia, il conosciuto Corrado Giustozzi, che da poco, e di questo tutti ne siamo fieri, è entrato a far parte della grande famiglia di ioProgrammo. Il sito a cui vi rimando che contiene un interessante motore di ricerca di anagrammi è *http://www .nightgaunt.org/anagrams*. Ma esaminiamo il metodo, cerchiamo di capire come si possano produrre anagrammi multiparola sulla base dei saperi acquisiti. Il primo passo da compiere è quello di considerare l'intera frase come un'unica parola, questa operazione di concatenazione delle singole componenti della frase si ottiene banalmente eliminando dalla stringa di partenza gli spazi, ottenendo in tal modo una sequenza di lettere. Di questa stringa bisogna calcolare la firma che indicheremo con *sorgente*. Si tratta adesso di ricercare anagrammi all'interno della firma sorgente. Per farlo si valuta la generica firma consultata nel dizionario omonimo, che chiameremo *oggetto*, e la si compara con la firma sorgente. Una generica firma rispetto alla sorgente può trovarsi in tre casi diversi.

Il primo, è che le due firme siano coincidenti, vorrà dire che siamo stati fortunati e che quindi abbiamo quanto meno trovato un anagramma, possiamo quindi procedere per la ricerca di altri anagrammi. Il secondo caso si ha quando la firma oggetto, ossia quella consultata nel dizionario delle firme, sia contenuta nella sorgente. Ciò porta a sperare che nella restante parte di stringa sorgente vi siano altre parole (firme) e che quindi si possa comporre la frase anagrammata. Quindi in tal caso si procede sottraendo alla firma sorgente quella contenuta e riapplicando lo stesso procedimento di ricerca anagrammi alla stringa così ottenuta, è evidente che la stringa oggetto debba essere accantonata momentaneamente da qualche parte. Riapplicare lo stesso procedimento significa richiamare la stessa routine che stiamo descrivendo passando questa volta non la firma sorgente ma la firma ottenuta dalla sottrazione tra la sorgente e la oggetto. Il terzo caso si ha quando la firma sorgente non è contenuta in quella oggetto, ovviamente in tale situazione si scarta la soluzione e si passa alla valutazione delle successive firme oggetto consultate nell'apposito dizionario. Merita uno specifico approfondimento il secondo punto la cui definizione svela la natura ricorsiva del procedimento. Si tratta, infatti, di applicare la stessa procedura man mano a stringhe di lunghezza minore fin quando non si verifica la condizione di uscita che può coincidere con il primo caso se viene trovato un anagramma multiparola o con il presentarsi più volte del terzo caso (esplorazione di tutte le possibili soluzioni) se siamo sfortunati e quindi non esiste alcuna frase anagrammata. La struttura ricorsiva attua un ricerca esaustiva di soluzioni che gli affezionati della sezione soluzioni di ioProgrammo conosceranno sicuramente, giacché mi capita di proporla periodicamente.

Mi sto riferendo ovviamente al backtracking. Si noti che la ricerca in profondità non sempre va a buon fine, può capitare che per diverse volte si trovi una firma oggetto nella sorgente o comunque nella firma derivata dalla sorgente per successive sottrazioni di stringhe oggetto e che si pervenga comunque ad un vicolo cieco. Può accadere cioè che applicando più volte il secondo caso non si trovi una stringa che sia uguale o contenuta nella sorgente, quindi bisogna fare un passo indietro (back) eliminare cioè l'ultima stringa oggetto e ricercare nuove soluzioni. Se anche questo caso non presenta soluzioni sarà necessaria un'altra retromarcia, secondo la ben conosciuta tecnica del backtracking. Solo per inciso ricordiamo che si presta ad essere gestita attraverso l'uso di uno stack con il quale si tiene traccia di tutte le chiamate a procedura con la push ed il rilascio delle stesse con la pop. Ad ogni modo i moderni compilatori implicitamente gestiscono la ricorsione con l'uso di uno stack di sistema. Ma queste sono tutte nozioni che ben possedete, per cui andiamo avanti analizzando ancora più a fondo il metodo per svelarne tutti i segreti e per poter poi sviluppare una struttura di algoritmo risolutore. Per fare il punto della situazione, deve essere chiaro che nel momento in cui l'algoritmo confronta la firma *oggetto* con la firma *sorgente* (per sorgente si intende anche quella che si ottiene man mano dalla sottrazione della prima firma sorgente con le successive oggetto prodotte) si possono verificare tre dei casi appena descritti.

Con il primo si esce e si ottiene una soluzione, semplicemente raccogliendo tutte le stringhe oggetto prodotte, si procede quindi automaticamente per la generazione di altre soluzioni riapplicando l'algoritmo a partire dalla firma immediatamente successive a quelle valutate. Con il secondo si procede in profondità con la ricerca e si generano le soluzioni parziali come stringhe oggetto. Infine, con il terzo si torna indietro, si effettua un back, e si passa alla procedura chiamante. La ricerca termina quando tutte le soluzioni possibili sono state esplorate, ovvero quando sono terminate le firme oggetto da valutare rispetto alle stringa sorgente iniziale.

Ovviamente, la ricerca può risultare infruttuosa quindi in alcuni casi può non produrre alcun anagramma multiparola.

Curiosità

Vi sono alcune curiosità di carattere storico geografico legate all'affascinante mondo degli anagrammi. Tra le più antiche si annovera un aneddoto, che sembra avere solidi riscontri storici, secondo il quale il re di Francia Luigi XIII aveva nominato e stipendiato una persona con il solo compito di produrre anagrammi. Da un punto di vista geografico è interessante notare che il Giappone è l'unico paese al mondo che ha importanti città che sono tra loro anagrammi Tokyo e Kyoto.

Varianti di anagrammi

I patiti dell'enigmistica hanno concepito nuove ed interessanti varianti degli anagrammi. Tra le più curiose vi è il panagramma che non è altro che un anagramma costituito da tutte le lettere dell'alfabeto (26 nel caso più generale di alfabeto anglosassone). Si definisce anigramma, ossia anagramma animato da anigram, un modo animato di mostrare la formazione dell'anagramma. Per intenderci gli anigrammi possono essere efficacemente prodotti con strumenti come macromedia flash. Per concludere la mini rassegna è interessante la figura dell'anugramma (traduzione di anugram) che è un particolare anagramma multiparola che non cambia il significato. Vi ripropongo l'esempio in inglese che ho trovato in giro per internet, sono sicuro che ve ne sono di sorprendenti anche in italiano:

Eleven plus two =
Twelve plus one

Insomma la somma fa sempre thirteen, tredici, 13. Da notare che, per una generica frase, gli anugrammi sono dei sotto insiemi degli anagrammi, ma non vale il contrario.

Soluzioni

ALCUNE IMPORTANTI CONSIDERAZIONI

L'output dell'algoritmo che ci apprestiamo a sviluppare secondo il procedimento appena descritto è una sequenza, a volte lunga, di "frasi". Ho usato le virgolette per specificare che non si tratta propriamente di frasi ma si ha a che fare con sequenze di parole tutte rigorosamente appartenenti al dizionario della lingua italiana. Chi però, pensava che il risultato finale fosse un insieme di frasi tutte di senso compiuto rimarrà un po' deluso. L'output prodotto va trattato manualmente in due modi. Premetto che è raro il caso in cui una delle frasi prodotte abbia una semantica corretta. Quindi il primo tentativo consiste nel valutare la sequenza di parole per capire se è possibile disporle in modo differente tale da dare un significato alla frase. Se il tentativo non porta i frutti sperati bisogna passare all'eliminazione della sequenza.

Permutazioni

PERMUTAZIONI SEMPLICI

Il numero di permutazioni di n elementi è pari al fattoriale di n:

Pn=n!
=n*(n-1)*(n-2)*..*1
P0=1

PERMUTAZIONI CON ELEMENTI RIPETUTI

Se le permutazioni presentano gli stessi elementi nella stessa configurazione, cioè se a è il numero di volte che si ripete un dato elemento, b le ripetizioni di un secondo elemento, c di un terzo e così via. Allora le permutazioni in cui sono presenti degli elementi ripetuti sono uguali a:

P(a,b,c,..)n=

Come esempio consideriamo un'urna contenente palline di colore: giallo, rosso e blu. Le Pn saranno:

grb, gbr, bgr, brg, rbg, rgb

che come suggerisce la formula sono sei. Si parla di permutazioni distinte con elementi ripetuti quando uno o più elementi possono riproporsi più volte. Se alcuni elementi si ripetono a,b,c .. volte allora indicheremo tali permutazioni come:

P(a,b,c,..)n

Ovviamente a parità di n le permutazioni con elementi ripetuti sono in numero minore. Dove n è il numero totale di elementi, computando cioè anche le ripetizioni. Se cioè si vuole fare l'anagramma della parola "pippo", n vale 5 e non 3.
Utili riferimenti sull'argomento si possono trovare nell'articolo della sezione Soluzioni ioProgrammo n. 18 – autore Fabio Grimaldi.

Con questa doppio accorgimento al termine della valutazione manuale otterremo il risultato sperato. Di seguito è riportato uno scorcio del output prodotto dal motore di ricerca sviluppato da Corrado Giustozzi a fronte del mio nome e cognome fornito come input.

```
...
154: dal fiori gambi
155: daro fili gambi
156: di fa gambi orli (fa orli di gambi)
157: di fara globi mi
158: di fari gambi lo
159: di fari gambo il
160: di faro gambi il
161: di faro gambi li
162: di fibra il mago (il mago di fibra)
163: di fibra li mago
164: di fibra maglio (maglio di fibra)
165: di fora gambi il
...
```

Le scritte tra parentesi sono state riportate da me manualmente per dare senso alla frase. È possibile sperare che in futuro l'output finale di un qualsiasi motore sia una frase di senso compiuto, ma il problema come gli esperti di linguistica automatica ben sanno è alquanto spinoso. Non a caso si riscontrano, ad esempio difficoltà nella traduzione automatica di testi da diverse lingue. Ma la trattazione si fa complessa, magari sarà oggetto di una futura discussione tra queste pagine.

L'ALGORITMO DI RICERCA

Facendo riferimento alla struttura dati ed alle routine presentate la scorsa volta, un'implementazione del procedimento già accuratamente descritto potrebbe essere la seguente. Non si tenga conto della presentazione ne tanto meno dell'efficienza. Lo scopo di questo paragrafo è descrivere il metodo di chiamata ricorsiva che innesca il backtracking. Cominciamo con quello che può essere definito *main program:*

```
// main program
inizio
scaricadizfirme(sorgfirme,firme)
leggi(frase)
sorgente <- calcolafirma(concatena(frase))
anagrammamulti(sorgente,destinazione)
scrivi(destinazione)
fine
```

Si caricano le firme, si legge la frase da anagrammare, la si concatena e si calcola la firma, poi si richiama la procedura di produzione di anagrammi multipli *anagrammamulti*.
L'output sarà riportato sul vettore *destinazione*

(ogni riga-cella una frase anagrammata)

```
// Proc anagrammamulti
inizio
i<-0
ripeti
  sorg<-sorgente
   i <-i+1
   j<-i
  valuta(firma[j], sorg)
finche i=max
scrivi(destinazione)
fine
```

Supponendo che tutte le firme siano presenti in un vettore si scorre questo ultimo al fine di comparare le singole firme con la stringa sorgente chiamata *sorg*. La procedura *valuta* è il fulcro del programma. Si controlla che l'indice delle firme non sia in overflow (*max* è il numero di firme), poi verifichiamo se ci troviamo nel caso 1 prima descritto, se così fosse avremmo trovato una soluzione e potremmo tornare nella procedura *anagrammamulti* alla ricerca di altre. Se la condizione precedente è falsa si controlla che la firma *j-esima* sia contenuta nella stringa parziale *sorg*. In tal caso si sottrae la stringa firma da *sorg*, si aggiunge al risultato parziale *oggetto* e si valuta ricorsivamente la stringa rimanente. Se invece la condizione non è vera prima di abbandonare la procedura chiamante per effettuare il backtracking si incrementa la *j* per poter valutare la prossima firma, automaticamente l'oggetto si libera dall'ultima soluzione (poiché si fa riferimento all'oggetto precedente).

```
// Proc valuta
inizio
se j<=max allora
  azzera(oggetto)
  se sorg=firme[j] allora
    oggetto<-aggiungi(oggetto,sorg)
    soluzione(destinazione, oggetto)
  altrimenti
  se contenuto(firme[j], sorg) allora
    sorg<-sottrai(firme[j],sorg)
    oggetto<-aggiungi(oggetto, sorg)
    valuta(firma[j],sorg)
  altrimenti
    j<-j+1
fine
```

Le procedure *aggiungi* e *sottrai* fanno rispettivamente una concatenazione e una estrazione di stringhe. Mentre *soluzione* aggiunge al vettore destinazione la stringa oggetto. I blocchi di codice vanno valutati per come sono intentati. L'algoritmo è migliorabile e personalizzabile come si può riscontrare nelle applicazioni presenti anche su internet che sviluppano motori di ricerca di anagrammi multiparola. Ad esempio, si possono applicare alcuni vincoli come il numero di parole della frase, la presenza o l'assenza forzata di specifiche parole e altro. Può essere uno stimolante esercizio pensare a come si possa modificare l'algoritmo per attuare tali estensioni.

CONCLUSIONI

Le applicazioni degli anagrammi non sono soltanto ludiche, vengono ad esempio usate in complessa materia di crittografia ed in altre applicazioni avanzate. Ad ogni modo spero che l'argomento sia stato gradito. Personalmente lo apprezzo molto perché è uno di quei casi, come altri che ho presentato tra queste pagine e che ricerco per studiarli ed eventualmente tradurli in articoli, in cui sono egualmente interessanti e sorprendenti sia il problema in se che la sua soluzione algoritmica. È arrivato il momento di congedarmi, la versione postmoderna di esperto di reti ad alta velocità è pronta per voi a ricercare nuove ed emozionanti soluzioni. Alla prossima!!

Il mago di fibra
Fabio Grimaldi

Backtracking

Gli algoritmi di backtracking sono anche conosciuti come tecniche di ricerca esaustiva della soluzione di problemi. Consistono in algoritmi che esplorano tutte le possibili soluzioni procedendo, per così dire, per tentativi. Nel caso più generale l'obiettivo finale viene diviso in sotto-obiettivi, ognuno di essi viene risolto in modo assestante, per cui può essere a sua volta partizionato in ulteriori problemi con corrispondenti sotto-obiettivi. Analizzando l'approccio alla soluzione si denota una logica ricorsiva. La ricerca della soluzione come esplorazione di sotto-obiettivi può essere interpretata come la visita di un albero di obiettivi dove i nodi sono appunto sotto-obiettivi ognuno dei quali può avere dei sotto-alberi come discendenti, ovvero altri sotto-obiettivi. Qualora la ricerca lungo un sotto albero risulti infruttuosa bisogna tornare indietro (back) ed esplorare altri rami. Questo parallelo "botanico", ci indica i pregi e i difetti del metodo, per cui se da un lato esplorando, nella peggiore delle ipotesi tutto l'albero, si perviene alla soluzione, è anche vero che l'albero può crescere secondo andamenti esponenziali rendendo di fatto la risoluzione alquanto costosa, in termini di complessità temporale; in alcuni casi il costo è così elevato da rendere la tecnica sostanzialmente impraticabile. Per ovviare questo ultimo aspetto negativo si rendono necessari metodi di potatura dell'albero che escludano a priori, secondo logiche proprie del particolare problema, alcune soluzioni.
Utili riferimenti sull'argomento si possono trovare negli articoli delle sezioni Soluzioni ioProgrammo nei numeri 12, 19, 26 e 27 – autore Fabio Grimaldi

XML e Flash

L'UTILE ED IL DILETTEVOLE SI INCONTRANO

XML e Flash

Esploriamo in questo articolo l'avvincente possibilità di interazione tra Macromedia Flash ed il protocollo XML, creando un'applicazione di esempio che usufruisca di tale tecnologia per implementare un menu dinamico, costruito interamente a partire da un file XML. Vedremo nel dettaglio il supporto che Flash offre per scambiare dati in questo formato, analizzando le funzioni più utilizzate dell'oggetto XML che Macromedia ha messo per noi a disposizione nel suo gioellino.

Oggigiorno, quasi tutto ciò che viene prodotto dall'industria del software riguarda in qualche maniera i dati e la loro elaborazione. Dal negozio virtuale al gestionale aziendale, oramai non vi è applicazione che non faccia uso di una base dati, sia questa un database relazionale o un semplice file di testo con la propria semantica. Esistono molti modi differenti per accedere, manipolare ed elaborare i dati e nuove tecnologie nascono ogni giorno per aggiungere funzionalità a quelle già esistenti: tra tutte, XML è quella che nel tempo ha riscosso maggior successo, sia per la sua semplicità che per la sua integrazione perfetta anche tra sistemi eterogenei. Un fattore importante per lo sviluppatore che si accinge ad utilizzare nel proprio software il protocollo XML, è la consapevolezza di trovare API o wrapper di supporto per la maggioranza dei linguaggi di programmazione e nella quasi totalità dei sistemi operativi; la sua struttura basata su markup, infatti, permette una facile implementazione di software che la interpreti, perlomeno per quanto riguarda le funzionalità più basilari. L'insieme di oggetti software (*Document Object Model* o DOM in breve) che permettono l'interazione con dati XML varia nella sua completezza di sistema in sistema e di produttore in produttore, e a seconda naturalmente delle necessità del software che ne farà uso. La maggior parte delle applicazioni che utilizzano XML, sfrutta il DOM messo a disposizione dal sistema operativo; Microsoft, ad esempio, sviluppa uno dei DOM XML più completi ed evoluti, tramite cui lavorare con flussi XML diventa un gioco da ragazzi. Basta dare un'occhiata su MSDN nella sezione dedicata a MSXML (questo è il nome in codice del prodotto) per farsi un'idea di quanto questo componente software sia potente. Gli sviluppatori di Macromedia Flash Player, vista la natura multi piattaforma che avrebbe avuto tale software, hanno preferito non affidarsi ai vari DOM dei sistemi operativi per i quali andavano sviluppando ma hanno optato per costruirsi un DOM proprietario, in comune tra tutte le implementazioni di player. Il supporto XML di Flash, reso disponibile solo a partire dalla versione 5.0, non offre molte funzionalità e, a prima vista, il suo funzionamento appare davvero criptico, anche (e oserei direi soprattutto) per chi già utilizza altri DOM XML; i vantaggi offerti, però, dallo studio di questo modello ad oggetti e dalla conseguente possibilità di elaborare dati XML tramite Flash sono molti e aprono le porte a progetti software completamente dinamici e di altissimo impatto visivo. Scopo di questo articolo, come già annunciato, è la realizzazione di un'applicazione Flash che costruisca dinamicamente un semplice menu a partire da un file XML che ne descrive interamente le funzionalità e la struttura: il risultato sarà completamente dinamico e indipendente dai dati. La qualità estetica dell'applicazione esula dall'obiettivo preposto, pertanto si lascia alla fantasia del lettore l'ampliamento di quest'ultimo punto.

IN PRINCIPIO FU XML

Prima attrice della nostra applicazione è senz'altro la base dati XML, impiegata come sorgente strutturale del menu che andremo a costruire. Seguendo la buona pratica, stabiliamo a priori lo schema seguito dalla struttura descrittiva del nostro menu; innanzitutto decidiamo di quali dati abbiamo bisogno per visualizzare gli elementi – ovvero le voci – di cui quest'ultimo è composto. Fortunatamente il nostro problema è molto semplice e possiamo tranquillamente convenire che per creare una voce di menu ci occorrono due dati fondamentali, un testo descrittivo della voce e un url verso cui puntare. Stabiliamo, inoltre, che entrambi i va-

XML e Flash

XML e Flash

L'utile ed il dilettevole si incontrano

lori saranno di tipo stringa. Definita così la struttura della singola voce, affermiamo ancora che a livello globale ciascuna voce può contenere a sua volta un numero n di sottovoci con caratteristiche uguali alla prima. Quest'ultima asserzione ci permette la costruzione dinamica di menu a più livelli.
Vediamo una realizzazione applicativa di quanto appena descritto:

```
<menu>
 <elementi>
  <voce testo="Edizioni Master" url=
                              "http://www.edmaster.it" />
   <elementi>
    <voce testo="Prodotti" url=
                "http://www.edmaster.it/?job=prodotti" />
     <elementi>
      <voce testo="ioProgrammo" url=
        "http://www.edmaster.it/?job=prodotti&amp;id=4" />
      <voce testo="Internet Magazine" url=
        "http://www.edmaster.it/?job=prodotti&amp;id=3" />
     </elementi>
    <voce testo="ITPortal" url="http://www.itportal.it/" />
   </elementi>
  <voce testo="Macromedia" url=
                        "http://www.macromedia.com" />
   <elementi>
    <voce testo="FlashMX" url=
         "http://www.macromedia.com/software/flash/" />
   </elementi>
  <voce testo="World Wide Web Consortium" url=
                                "http://www.w3c.org" />
 </elementi>
</menu>
```

Si osservi la struttura gerarchica descritta dal documento appena esposto: un tag radice, *menu*, raggruppa l'intero insieme di dati. Con ordine, caratteristica fondamentale per la buona riuscita di qualsiasi sistema software, vengono raggruppate tutte le voci di menu del primo livello sotto il tag *elementi*, appartenente al tag *radice*. Al di sotto di *elementi*, vengono esposti i tag *voce*, che incapsulando gli attributi *testo* e *url* danno vita alle voci di menu di primo livello. Gli elementi *voce*, come è possibile osservare chiaramente nell'esempio proposto, possono contenere essi stessi il tag *elementi*, che a loro volta possono contenere tag di tipo *voce*, così da permettere, come si annunciava, la possibilità di creare menu a livelli multipli.
Nel nostro fine applicativo, continuiamo lo sviluppo del menu e diamo nome al file appena creato *Menu.xml*; una volta salvato il file, lo rendiamo disponibile per la lettura da parte di Flash, salvandolo nella stessa cartella dove risiederà l'swf che andremo a costruire nel resto dell'articolo. Si tenga presente che Flash, se utilizzato all'interno di una pagina html sotto forma di plugin, consente di default l'apertura di file XML che sono residenti solo all'interno dello stesso sottodominio dell'swf caricato: nella maggioranza dei casi ciò significa che il plugin di Flash non consente di caricare dati da un sito web che non sia quello da cui viene lanciato l'swf. A tale limitazione, studiata in effetti per non essere tale ma per offrire un livello di sicurezza maggiore agli utenti del plugin, va incontro anche la funzione *LoadVariables()*, così come il nuovo oggetto *LoadVars* di FlashMx, che altro non è che un wrapper di quest'ultima.

Fig. 1: Il menu XML appena creato, aperto tramite Internet Explorer.

Non soffrono di questo limite, invece, gli swf non inglobati in pagine html ma aperti in maniera diretta tramite il player apposito (solitamente utilizzato dai soli sviluppatori Flash).

L'INTERFACCIA

Come già anticipato, la porzione del nostro applicativo che opererà con l'utente utilizzerà unicamente l'interfaccia Flash 5 (e l'html necessario per istanziare il plugin swf). Le operazioni che affidiamo a questo modulo sono a livello logico abbastanza semplici; sostanzialmente utilizzeremo actionScript per caricare la struttura XML appena creata e per visualizzare a video il nostro menu, utilizzando quest'ultima come direttrice. Per chiarezza, dividiamo il codice che andremo a sviluppare in tre step: acquisizione dei dati, elaborazione dei dati e visualizzazione a video. Creiamo quindi il movie con cui lavoreremo nel resto dell'applicazione di esempio. Non sono importanti qui le impostazioni grafiche, lascio al lettore il piacere di procedere verso questa direzione come meglio crede. Il codice actionScript che andremo a creare nel resto dell'applicazione risiederà pertanto per semplicità nel primo e unico frame del nuovo movie. Con la convinzione che effettuare una buona progettazione ora servirà a facilitarci le cose in un secondo momento, procediamo a scendere nel dettaglio dell'implementazione degli step di cui si accennava poche righe fa. Non nascondo che la difficoltà si abbatte drasticamente se la base dati è ben predisposta: fortunatamente quest'ultima è stata creata da zero ed ho preferito dare ad essa una struttura chiara e facile. Di conseguenza i primi due step – acquisizione dei dati e elaborazione dei dati – non ci creeranno troppi grattacapi. L'acquisizione dei dati avviene in maniera pressochè automatica; eccone il codice:

```
1 var xmlMenu = new XML();
```

Dom

L'insieme di oggetti software *(Document Object Model o DOM in breve)* che permettono l'interazione con dati XML varia nella sua completezza di sistema in sistema e di produttore in produttore, e a seconda naturalmente delle necessità del software che ne farà uso.

XML e Flash

XML e Flash
L'utile ed il dilettevole si incontrano

```
2
3 xmlMenu.ignoreWhite = true;
4 xmlMenu.onLoad = MenuLoaded;
5 xmlMenu.load('Menu.xml');
```

In questa prima porzione di codice avviene la creazione dell'oggetto *XML* e del caricamento dei dati presenti nel menu XML, creato poco fa. Andando nel dettaglio viene effettuata la dichiarazione e la creazione dell'oggetto *xmlMenu* (1), viene impostata la proprietà *ignoreWhite* a *true* (2), associato all'evento *onLoad* la funzione *MenuLoaded* che tra poco analizzeremo (3) e infine caricato il file *menu.xml* (4). Fino a qui non possiamo ancora determinare se tale file è stato caricato o meno nell'oggetto *xmlMenu*. Per poterlo fare è necessario entrare nel dettaglio della funzione che viene lanciata non appena il caricamento del file XML è terminato.

```
10 function MenuLoaded(bSuccess)
11 {
12  if(!bSuccess) {
13   trace('Menu non caricato.');
14   return;
15  }
16
17  trace('Menu caricato.');
18
19  var ndRoot = xmlMenu.firstChild;
20  var ndElementi = ndRoot.firstChild;
21
22  DisplaySubMenu(ndElementi, 0);
23 }
```

Come anticipato, *MenuLoaded* viene richiamata quando il caricamento del file XML termina. È possibile sapere se tale caricamento è andato a buon fine analizzando il parametro *bSuccess*, booleano, valorizzato a *false* in caso di problemi, a *true* altrimenti. Come è possibile osservare, il codice d'esempio effettua un trace dichiarando lo stato dell'operazione (13 e 17) e in caso questa non sia stata andata a buon fine esce dalla funzione (14) annullando di fatto l'intera operazione. Nell'ipotesi in cui *menuXML* contenga la struttura presente nel file *menu.xml*, vengono dichiarate due variabili. La prima, *ndRoot* (19), è in realtà il primo elemento presente all'interno della struttura XML, ciò che nella totalità dei DOM XML presenti nel mercato viene chiamato *nodo radice*; per Flash tale nodo è dunque il primo nodo figlio del documento XML caricato. Vista la struttura del documento XML, è conveniente creare una funzione generica (*DisplaySubMenu*), incaricata di elaborare in maniera ricorsiva i dati presenti nel tag *elementi*. Tale funzione visualizzerà i tag *voce* presenti nella struttura che verrà passata in parametro e richiamerà sé stessa qualora vi fossero altre strutture elementi figlie di quella originale. L'idea della ricorsione è molto utile in casi come questi dove le operazioni da effettuare per un blocco di dati sono identiche per *n* blocchi di uguale struttura. Data la necessità di elaborazione del tag *elementi*, richiediamo al nodo radice la referenza alla struttura di primo livello di questo tipo e associamo alla variabile *ndElementi* tale referenza. Valorizzata *ndElementi*, richiamiamo la funzione *DisplaySubMenu* di cui ora verrà fatta analisi approfondita.

```
30 var verticalDeep = 0;
31
32 function DisplaySubMenu(xmlData, deep)
33 {
34  var ndVoce = xmlData.firstChild;
35  while(ndVoce)
36  {
37   switch(ndVoce.nodeName)
38   {
39    case 'voce':
40     DisplayMenuItem(ndVoce, deep);
41     verticalDeep++;
42     break;
43    case 'elementi':
44     DisplaySubMenu(ndVoce, deep + 1);
45     break;
46   }
47
48   ndVoce = ndVoce.nextSibling;
49  }
50 }
```

DisplaySubMenu è la funzione cuore della fase dell'elaborazione dei dati. Come è possibile osservare nella sua dichiarazione (32), sono previsti due parametri di chiamata. Il primo, *xmlData*, sarà un nodo XML di tipo *elementi*. Il secondo parametro, *deep*, altro non sarà che la profondità orizzontale del submenu, in altre parole il livello del menu che la funzione andrà a creare. Non a caso nella chiamata effettuata in *MenuLoaded* (22), *deep* è impostato a zero. Alla stessa maniera viene mantenuta una variabile globale, *verticalDeep* (30), inizializzata a zero che altro non è che la profondità verticale del menu: servirà nella fase di visualizzazione per disegnare le voci di menu una sotto l'altra. La funzione crea dapprima la variabile *ndVoce* e la fa puntare al primo nodo figlio del nodo elementi ricevuto come parametro (34). Viene poi iniziato un ciclo che durerà fintanto che *ndVoce* sarà un nodo valido (35). Successivamente, se *ndVoce* è un tag di tipo *voce* (39) questo viene passato alla funzione *DisplayMenuItem* (40) che si occuperà di visualizzare la voce a video (la vedremo in seguito); se invece tale nodo è di tipo *elementi* (43), viene applicato il concetto di ricorsione già anticipato per cui viene chiamata nuovamente *DisplaySubMenu* (44), valorizzando questa volta il parametro deep ad un valore incrementato di uno rispetto al deep originale.
Infine *ndVoce* viene valorizzata con il suo nodo successivo dello stesso livello (48): qualora questo nodo non esistesse (al termine della struttura XML), *ndVoce* verrà

XML e Flash

Il supporto XML di Flash, reso disponibile solo a partire dalla versione 5.0, non offre molte funzionalità e, a prima vista, il suo funzionamento appare davvero criptico, anche (e oserei direi soprattutto) per chi già utilizza altri DOM XML; i vantaggi offerti, però, dallo studio di questo modello ad oggetti e dalla conseguente possibilità di elaborare dati XML tramite Flash sono molti e aprono le porte a progetti software completamente dinamici e di altissimo impatto visivo.

impostata automaticamente a *null*, di fatto interrompendo il ciclo. Va altresì aggiunto che *verticalDeep* viene incrementata ad ogni visualizzazione di voce di menu (41), proprio al fine di tenere traccia della profondità verticale nelle fase di presentazione dei dati. Al fine di ottimizzare lo sfruttamento delle risorse e migliorare la qualità del nostro lavoro, risulta ora conveniente prevedere la costruzione di un movieclip da utilizzare come modello di visualizzazione delle voci di menu. Tale movieclip verrà inserito nella library del movie su cui stiamo lavorando e ci permetterà di riutilizzarlo per costruire le *n* voci di menu di cui necessitiamo; la funzione di visualizzazione si occuperà dunque di duplicare questo movieclip e di impostarne le proprietà necessarie affinchè ogni voce contenga i dati corretti. Creiamo dunque preventivamente il movieclip *mvVoce*, dotiamolo di un testo dinamico al quale associeremo la variabile *Testo*. Fatto questo importiamolo nella library, ricordando di spuntare nelle proprietà di linkage la checkbox *Export for ActionScript:* tale impostazione ci permetterà di lavorare col movieclip presente nella libreria anche tramite il codice che andremo a creare. Finiti i preparativi, vediamo dunque nel dettaglio la funzione di visualizzazione delle voci di menu.

```
60 function DisplayMenuItem(xmlData, deep)
61 {
62  attachMovie('mvVoce', 'dmvVoce' + verticalDeep,
                                        verticalDeep);
63  var mvCurrent = eval('dmvVoce' + verticalDeep);
64
65  mvCurrent.onRelease = function() {
66   getURL(xmlData.attributes.url);
67  }
68
69  mvCurrent.Testo = xmlData.attributes.testo;
70  mvCurrent._x = 20 * deep;
71  mvCurrent._y = (mvCurrent._height + 2) *
                                        verticalDeep;
72 }
```

Come si è già potuto osservare, *DisplayMenuItem* viene richiamata con il parametro *xmlData* che punta ad un nodo voce della nostra struttura XML ed il parametro *deep* che indica il livello corrente del menu (ma anche il suo scostamento orizzontale, nel nostro caso). Inizialmente la funzione crea un nuovo *movieclip* (62), duplicando *mvVoce*, presente nella library. A questo duplicato viene assegnato un nome, creato dinamicamente, formato dalla stringa dmvVoce (la *d* sta per dinamico) e dalla profondità verticale che cambia ad ogni chiamata. *Eseguita* la duplicazione, viene assegnata alla variabile *mvCurrent* la referenza a tale nuovo *movieclip* (63), allo scopo di impostarne le proprietà nelle righe successive. Tramite la referenza appena creata, impostiamo la risposta all'evento *onRelease* del *movieclip* (65): trattandosi di una voce di menu, non facciamo altro che eseguire un *getURL* verso il collegamento specificato dall'attributo *url* del nodo XML *voce* preso in esame (66). Impostiamo inoltre la variabile *Testo*, associata al testo dinamico creato in precedenza all'interno del movieclip *mvVoce*: il nuovo valore sarà quello dell'attributo *testo* del nodo *voce corrente* (69). Non ci resta che impostare le coordinate entro cui il nuovo movieclip sarà disegnato; impostiamo dunque l'attributo *_x* ad un valore di scostamento a piacere (nell'esempio si è scelto 20) che ovviamente andrà moltiplicato per la profondità orizzontale del menu (70).

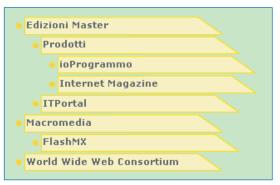



Fig. 2: **L'interfaccia Flash, dopo un piccolo ritocco grafico.**

Infine impostiamo l'attributo *_y* pari all'altezza del movieclip più un valore di scostamento a piacere (nell'esempio 2), moltiplicando il tutto questa volta per la profondità verticale. Il motore di visualizzazione del nostro menu XML è ora pronto e funzionante; sarà sufficiente solo qualche accorgimento grafico per inglobarlo nelle vostre soluzione dinamiche basate su tecnologia Flash.

CONCLUSIONI

Le possibilità offerte dall'elaborazione dei dati tramite il motore XML fornito da Flash sono notevoli; nonostante il supporto incompleto che questo prodotto offre per la manipolazione dei dati secondo tale modello, è comunque facile creare applicazioni completamente dinamiche che utilizzano l'interfaccia Flash. L'interesse di Macromedia verso questa tecnologia sembra comunque essere in continuo aumento: se fino alla versione 5 il motore XML era codificato internamente tramite actionScript (con l'enorme penale della velocità di esecuzione), con la versione MX di Flash la casa produttrice ha scelto di optare per una compilazione in C++, inglobando il supporto XML nello stesso motore del player.

Nella visione globale del web, sia le componenti server che quelle client tendono sempre più verso la tecnologia XML, facendone di fatto uno standard verso cui guardare con fiducia. Non ci rimane che sfruttare al meglio ciò che il mercato ci propone, seguendo questa tendenza che, come abbiamo visto nel corso dell'articolo, offre ampi spunti di riflessione per mettere all'opera la fantasia.

Efran Cobisi

XML e Flash

XML e Flash

L'utile ed il dilettevole si incontrano

XSLT

Il linguaggio XSLT (*eXtensible Stylesheet Language Transformations*), nasce dall'esigenza di definire uno strumento generale che consenta di trasformare e manipolare un documento XML. Così come per l'XML, anche l'XSLT prende vita dal consorzio W3C, organo che si occupa della sua standardizzazione. La versione corrente di XSLT è la 1.0 (raccomandazione W3C del 16 Novembre 1999), al momento della stesura di questo testo è stata proposta la versione 1.1, attualmente reperibile come working draft
http://www.w3.org/TR/xslt11/

XML e Flash

XML e Flash
L'utile ed il dilettevole si incontrano

L'OGGETTO XML DI ACTIONSCRIPT PRONTUARIO DELLE FUNZIONI PIÙ USATE

new XML([sorgente])

È il costruttore dell'oggetto XML, vale a dire il metodo da richiamare per creare un oggetto di questo tipo. Accetta opzionalmente come unico parametro una stringa di testo XML con cui istanziare il nuovo oggetto.

Es: Creazione di un nuovo oggetto XML vuoto

```
var xOggetto = new XML();
```

Es: Creazione di un nuovo oggetto XML con alcuni dati

```
var xOggetto = new XML('<provincia>PD</provincia>
                        <provincia>MI</provincia>');
```

PROPRIETÀ

attributes

È un array associativo che contiene gli attributi xml di un particolare nodo.

Es: Accesso ad un attributo xml

```
var xOggetto = new XML('<provincia sigla="PD"
             descrizione="Padova" /><provincia sigla="MI"
                                 descrizione="Milano" />');
trace(xOggetto.firstChild.attributes.sigla);

// Visualizza in output la stringa PD
```

childNodes

È un array che contiene i nodi figli di un particolare nodo xml.

Es: Accesso al secondo nodo figlio di una struttura XML

```
var xOggetto = new XML('<provincia sigla="PD"
   descrizione="Padova" /><provincia sigla="MI" descrizio-
                                         ne="Milano" />');
var xFiglio = xOggetto.childNodes[1];
```

firstChild

Proprietà di un qualsiasi oggetto XML che punta al primo nodo figlio di quest'ultimo. È conseguentemente una scorciatoia per *oggetto.childNodes[0]*. Nel caso in cui l'oggetto XML non abbia nodi figlio, tale proprietà ritorna *null*.

lastChild

Proprietà di un qualsiasi oggetto XML che punta all'ultimo nodo figlio di quest'ultimo. Anche qui, nel caso in cui l'oggetto XML non abbia nodi figlio, tale proprietà ritorna *null*.

ignoreWhite

Proprietà booleana che indica, se impostata a *true*, di non prendere in considerazione gli spazi e i ritorni di carrello presenti tra un tag ed un altro. È molto utile in tutti quei casi dove la struttura xml che si va a leggere contiene questi tipi di caratteri e ciò è molto comune, specie se il vostro file XML è chiaro ed ordinato.
Di default è impostata a *false*, quindi Flash interpreta in maniera scorretta tali tipi di file. È consigliabile impostarla di regola sempre a *true* prima di procedere alla lettura di qualsiasi struttura XML.

loaded

Proprietà booleana che indica il corretto caricamento della struttura XML in cui viene chiamata. Se ritorna *true*, il documento è stato caricato con successo.

Es: Controllo sullo stato di caricamento dell'oggetto XML preventivamente creato

```
trace(xOggetto.loaded);
// Se xOggetto è stato caricato correttamente visualizza
                                            true in output.
```

nextSibling

Ritorna l'oggetto XML successivo nello stesso livello corrente, appartenente allo stesso nodo genitore. Questa proprietà è utile quando si intende effettuare la scansione di un intero set di nodi XML appartenenti ad uno stesso nodo genitore.

Es: Stampa in output tutti gli oggetti figlio di un particolare nodo XML

```
var xOggetto = new XML('<provincia sigla="PD" />
        <provincia sigla="MI" /><provincia sigla="RM" />');
var xFiglio = xOggetto.firstChild;
while(xFiglio) // Questo ciclo si ferma se xFiglio è null
{
  trace(xFiglio.attributes.sigla); // Visualizzo di ogni
                                  elemento l'attributo sigla
  xFiglio = xFiglio.nextSibling;
}
// Visualizza PD MI RM
```

previousSibling

Ritorna l'oggetto XML precedente nello stesso livello corrente, appartenente allo stesso nodo genitore. Questa proprietà è utile quando si intende effettuare la scansione di un intero set di nodi XML appartenenti ad uno stesso nodo genitore. La funzionalità è inversa rispetto a quella offerta da *nextSibling*.

Es: Stampa in output tutti gli oggetti figlio di un particolare nodo XML (scansione all'indietro)

```
var xOggetto = new XML('<provincia sigla="PD" />
        <provincia sigla="MI" /><provincia sigla="RM" />');
var xFiglio = xOggetto.lastChild;
while(xFiglio) // Questo ciclo si ferma se xFiglio è null
{
  trace(xFiglio.attributes.sigla); // Visualizzo di ogni
                                  elemento l'attributo sigla
  xFiglio = xFiglio.previousSibling;
}
// Visualizza RM MI PD
```

XPath

Il linguaggio XPath viene utilizzato per formulare espressioni in grado di valutare ogni nodo di un documento xml e di svolgere determinate operazioni su di essi, restituendo a particolari valori di tipo stringa, numerici o booleani. Il metodo con il quale XPath esplora le diverse strutture di nodi, all'interno del documento XML, è denominato percorso di posizione. Un percorso di posizione dispone di una sintassi specifica che consente di includere operatori ed espressioni utilizzati nell'individuazione di parti di un documento in base al tipo di struttura che rappresentano.

PROPRIETÀ

nodeName

Proprietà che ritorna il nome del tag del nodo XML corrente.

Es: Stampa in output del nome del tag del primo nodo presente in una struttura XML

```
var xOggetto = new XML('<provincia sigla="PD" /><provincia
sigla="MI" /><provincia sigla="RM" />');
var xFiglio = xOggetto.firstChild;
trace(xFiglio.nodeName);
// Visualizza provincia
```

nodeType

Proprietà che ritorna il valore 1 se il nodo XML corrente è un elemento, il valore 3 se il nodo è un nodo contenente solo testo. Si noti che Flash tratta i nodi testuali come unici nodi figlio del nodo che li contiene.

Es: Stampa in output del tipo di nodo per due nodi XML

```
var xOggetto = new XML('<provincia sigla="PD">
    <commento>Si trova in Veneto</commento></provincia>');
trace(xOggetto.firstChild.nodeType); // provincia
trace(xOggetto.firstChild.firstChild.firstChild.nodeType);
                         // Nodo testuale all'interno di commento
// Visualizza 1 3
```

nodeValue

Proprietà utile solo per i nodi di tipo testuale (*nodeType* è uguale a 3). Ritorna il testo presente all'interno del nodo.

Es: Stampa in output del valore di un nodo testuale

```
var xOggetto = new XML('<provincia sigla="PD">
    <commento>Si trova in Veneto</commento></provincia>');
trace(xOggetto.firstChild.firstChild.firstChild.nodeValue);
                         // Nodo testuale all'interno di commento
// Visualizza Si trova in Veneto
```

parentNode

Ritorna il nodo genitore di un particolare nodo XML. Se tale nodo XML non appartiene ad alcun nodo genitore, la proprietà ritorna null.

status

Proprietà che indica se un documento XML è stato caricato nell'oggetto XML con successo. Ritorna un valore interpretabile come segue:

0	Nessun errore
-2	Una sezione CDATA non è stata chiusa correttamente
-3	La dichiarazione XML non è corretta
-4	La dichiarazione DOCTYPE non è corretta
-5	Un commento non è stato chiuso correttamente
-6	Un elemento XML non è corretto
-7	Memoria esaurita
-8	Un valore di attributo non è stato chiuso correttamente
-9	Un tag iniziale non è stato chiuso con un tag finale
-10	Un tag finale non è stato aperto da un tag iniziale

METODI

load(url)

Carica il documento XML specificato nel parametro *url*. Tale documento deve risiedere nello stesso sottodominio del movie Flash chiamante. A caricamento completato si occupa di impostare il valore di loaded pari a true e scatena l'evento *onLoad()*.

Es: Caricamento di un file XML

```
var xOggetto = new XML();
xOggetto.Load('http://localhost/file.xml');
```

parseXML(testoXML)

Carica la struttura XML specificata nel parametro testoXML. Tale struttura andrà a sovrascrivere quella precedente (se esistente). La sua funzione è analoga a quella offerta dal costruttore della classe.

Es: Caricamento di una struttura XML

```
var xOggetto = new XML();
xOggetto.parseXML('<provincia sigla="PD"><commento>
                Si trova in Veneto</commento></provincia>');
```

hasChildNodes()

Ritorna un valore booleano che indica se il nodo corrente ha nodi figlio o meno.

getBytesLoaded()

Ritorna il numero di byte caricati per il documento corrente. Utile se utilizzata in coppia con *getBytesTotal()* per fornire percentuali di caricamento dei documenti XML.

getBytesTotal()

Ritorna la dimensione in byte del documento corrente.

EVENTI

onLoad(bSuccess)

Evento scatenato dalla funzione *Load()* quando il documento XML da caricare viene caricato. Il parametro che le viene passato (*bSuccess*) è di tipo booleano e indica se il documento è stato caricato con successo (*true*) o no (*false*).

Es: Caricamento di una documento XML dal web

```
var xOggetto = new XML();
xOggetto.onLoad = function(bSuccess)
{
    if(bSuccess)
    { trace('Documento caricato con successo.'); }
    else
    { trace('Documento non caricato!'); }
}
xOggetto.Load('http://localhost/file.xml');
```

XML e Flash

XML e Flash

L'utile ed il dilettevole si incontrano

I percorsi di posizione

I percorsi di posizione rappresentano il più importante strumento per generare espressioni XPath. Un percorso di posizione consente di avere accesso ai nodi, sia nello specifico del contesto, sia indipendentemente dal contesto; possiamo formalizzare un percorso di pozione con un'espressione che identifica un insieme di nodi dell'albero, ognuno di questi noto come node-set.

PASSWORD
A "PORTATA DI MANO"

Prendendo spunto da quanto discusso la scorsa volta sulla sicurezza di Windows e sui meccanismi di autenticazione, parliamo questa volta delle procedure di autenticazione biometriche, basate sul riconoscimento delle impronte digitali.

Nel numero precedente di ioProgrammo, si è ampiamente parlato della sicurezza di Windows 2000/XP e si è analizzato, da vicino, il meccanismo di autenticazione implementato da Microsoft per il suo sistema operativo, illustrando come fosse possibile modificare il modulo di login (chiamato GINA), per crearne uno personalizzato. La personalizzazione del sistema di accesso di Windows è oggi spinta al limite dalle aziende informatiche che si occupano di Information Security, che spesso accoppiano le tecnologie da noi trattate nel precedente articolo con i dispositivi biometrici per realizzare meccanismi di autenticazione che possono fare a meno delle password!

I SISTEMI BIOMETRICI

L'autenticazione e il riconoscimento dell'identità nel mondo informatico, oggi, sono argomenti che suscitano notevole interesse, grazie soprattutto all'espansione dei dispositivi biometrici (il cui costo è ormai alla portata di tutti) e a fronte delle continue richieste nei settori commerciale/militare. Proprio per questo motivo ioProgrammo si è sentita in obbligo di approfondire queste tematiche, cercando, con questo articolo, di trattare alcuni aspetti delle tecnologie biometriche. Stando alla definizione classica, un generico sistema di autenticazione può essere classificato in una di queste tre categorie, contraddistinte dalle diverse tecniche utilizzate:

- autenticazione mediante dati/informazioni che una persona conosce (password/PIN);
- autenticazione mediante oggetti/dispositivi che una persona possiede (chiavi fisiche come le smartcard);
- autenticazione mediante caratteristiche fisiche uniche dell'interessato (impronte digitali, voce, retina);

La scienza della biometria, nata sul finire degli anni 1950, si occupa proprio di quest'ultima categoria, cioè quella che tratta l'autenticazione e il riconoscimento effettuato mediante la rilevazione delle caratteristiche fisiche individuali di ciascuna persona, realizzato – ovviamente – con l'ausilio dei computer. In questo articolo ci occuperemo di un ramo particolare della biometria, quello che studia le impronte digitali, discutendo dapprima il ruolo dei sistemi biometrici nel mondo informatico e mostrando, infine, un esempio di applicazione biometrica capace di schedare (e successivamente identificare) le persone, mediante un apposito scanner di impronte digitali, gentilmente concesso in uso, per i nostri esperimenti, dalla Eutron (*www.eutron.it*), società leader nel settore dell'Information Security. I requisiti necessari per una corretta lettura degli argomenti trattati sono ridotti al minimo: le nozioni essenziali riguardo le tecniche biometriche saranno esposte nella parte introduttiva dell'articolo, mentre per la comprensione del codice è richiesta la sola conoscenza di Visual Basic e dell'uso delle API; naturalmente la compilazione e l'uso dei sorgenti di esempio necessitano di un sensore di impronte digitali come quello prodotto dalla Eutron e dell'apposito SDK.

PERCHÉ UNA PASSWORD NON BASTA?

Perché non basta usare semplicemente una password? E' una domanda che molti si pongono, in maniera del tutto legittima, e alla quale si può rispondere con qualche piccola statistica prelevata dai documenti del consorzio per le tecnologie biometriche (*www.biometrics.org*). Le password "costano" troppo, non garantiscono un'adeguata sicurezza e sono poco pratiche.
Qualche dato indicativo testimonia che ciò che stiamo dicendo è vero:

- la maggior parte di telefonate (30-50%) ai centri di assistenza sono dovute a problemi degli utenti con le password;
- i blocchi dei sistemi dovuti a password dimenticate causano notevoli perdite in produttività e a volte anche perdite di dati;
- mediamente un hacker è in grado indovinare il 30% delle password di una rete aziendale mediante appositi tool;

Biometria

File sul CD
\soft\codice\FPDemoVB.zip

GINA

GINA è l'abbreviazione di *Graphical Identification and Authentication* e rappresenta il modulo di Windows 2000/XP responsabile dell'autenticazione e dell'accesso degli utenti. Tale modulo è localizzato nella libreria di sistema *MSGINA.DLL* ed è rimpiazzabile con altri moduli che ad esempio fanno uso di tecnologie biometriche, smartcard, chiavi hardware. Per ulteriori approfondimenti su GINA fare riferimento al numero 65 di ioProgrammo.

Biometria

Password
a "portata di mano"

- la lunghezza minima di una password "sicura" cresce col tempo e con l'aumentare della velocità di elaborazione dei calcolatori;
- l'utilizzo nella vita quotidiana di molti dispositivi elettronici obbliga gli utenti a ricordare troppe password e codici numerici;
- la maggior parte delle password viene spesso trascritta dagli utenti su file in chiaro, su fogli di carta e su Post-it, causando una consistente riduzione della sicurezza;

Al contrario, le tecnologie di riconoscimento biometrico presentano molti più vantaggi rispetto alle tradizionali password: l'identificazione è univoca, non può essere condivisa da più utenti (cosa che invece avviene con le password, che spesso sono infatti scambiate tra colleghi e amici!) e non si è costretti a ricordare alcuna cosa, né a portar dietro dispositivi fisici di accesso che potrebbero essere rubati. L'unico svantaggio di queste tecnologie era rappresentato dal costo eccessivo: dico "era" perché oggi il prezzo dei sensori biometrici è diventato talmente contenuto, da essere oramai accessibile a molte fasce di utenti.

Tecnologie Biometriche

Le tecnologie biometriche fino ad oggi implementate e utilizzate nel mondo informatico sono basate su :

- **Riconoscimento impronte digitali;**
- **Riconoscimento facciale;**
- **Riconoscimento della voce;**
- **Riconoscimento della geometria della mano;**
- **Scansione della retina e dell'iride;**
- **Riconoscimento della firma.**

Si differenziano tra loro per caratteristiche come l'affidabilità, il costo e la praticità di uso. Per quanto riguarda il riconoscimento facciale, si è ampiamente parlato di questo argomento nel numero 56 di ioProgrammo.

IL PROCESSO BIOMETRICO

Un sistema biometrico per calcolatori prevede in genere tre fasi distinte di lavorazione :

- Enrollment
- Verification
- Identification

La prima fase è anche detta di "training" ed è necessaria per memorizzare l'identità di una persona e per raccogliere i dati biometrici che consentiranno ad un computer in futuro di identificarla correttamente. È la fase più delicata del processo, perché è qui che si crea il modello (*template*) rappresentativo di una persona: un cattivo modello può indurre in errore la macchina, causando false identificazioni (un utente potrebbe spacciarsi per un altro) o il rigetto dell'identità reale (l'utente legittimo non viene autorizzato perché non riconosciuto). La fase di verifica è invece detta *"1-to-1 matching"*, ossia corrispondenza 1-a-1. E' la fase in cui un utente dall'esterno fornisce le sue credenziali biometriche (impronte, voce, retina, ecc.) per consentire al calcolatore di confrontarle col template memorizzato in precedenza, in modo da avere un riscontro: in sostanza il computer si pone la domanda *"sei veramente la persona che dici di essere?"*. Il processo di identificazione è simile a quello di verifica, ma lavora su un set di credenziali più esteso: in questo caso si parla di *"1-to-N matching"*, cioè la macchina non conosce l'identità di un utente e prova a scoprirla analizzando tutti i template memorizzati in un database, cercando quello più somigliante fra tutti, se esiste.

Fig. 1: STRUTTURA DI UN IMPRONTA
L'elemento fondamentale di ogni impronta è la "ridge line", ossia la cresta, che forma intrecci, biforcazioni, spirali e divergenze. Un disegno formato da più creste è detto "ridge pattern".

LE IMPRONTE DIGITALI

La scienza delle impronte digitali è antica e moderna allo stesso tempo, nel senso che la sua introduzione risale a molti secoli prima (la scoperta delle impronte è attribuita a Marcello Malpigli, nel 1700 circa) ma l'uso effettivo dei primi sistemi di riconoscimento a impronte è datato solo intorno al 1950-1970, ad opera dell'FBI. Le caratteristiche che rendono forte l'uso delle impronte digitali sono l'immutabilità (la configurazione delle impronte è sempre identica nel tempo e a tutte le età) e l'unicità (la probabilità di trovare due impronte uguali è minore di 10-20 anche per gemelli omozigote), senza contare altri fattori come la praticità e l'universalità del sistema. La classificazione delle impronte, in apparenza caotica, segue invece uno schema ben preciso, che fa uso di termini tecnici utili per individuare e catalogare le forme e i disegni rilevati su un'impronta: l'elemento fondamentale di un'impronta è la cresta *(ridge line)*, che può formare intrecci, biforcazioni, spirali e divergenze; un disegno formato da più creste è detto *ridge pattern*. Il concetto fondamentale su cui si basa l'identificazione tramite impronte digitali sono le *minutiae* (o minuzie), cioè quei punti significativi dove le creste terminano o si biforcano. Generalmente basta un insieme di 30-40 minuzie per avere un'identificazione corretta e con basso margine d'errore, infatti il matching fra due impronte viene fatto proprio mediante l'analisi della posizione e dell'orientamento di un certo set di minuzie. Può facilmente accadere che impronte digitali diverse possano avere in comune fra loro qualche minuzia, ma solo l'impronta originale raggiungerà un numero massimo di corrispondenze di minuzie; questo fatto permette inoltre di stabilire un valore di soglia nel riconoscimento, utile per aumentare/abbassare la sicurezza del riconoscimento. L'identificazione delle impronte fatta dai computer si basa sul calcolo dell'immagine direzionale dell'impronta digitale: l'immagine viene dapprima ac-

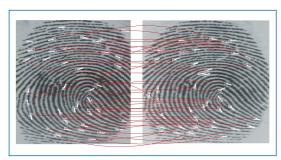



Fig. 2: MATCHING E MINUTIAE
L'identificazione delle impronte digitali si basa sulle minutiae (o minuzie), punti significativi dove le creste terminano o si biforcano. Il matching fra due impronte viene fatto proprio mediante l'analisi della posizione e dell'orientamento di un certo set di minuzie (30-40 sono sufficienti).

quisita dagli appositi sensori CCD/CMOS che usano lenti particolari per rilevare le impronte, quindi viene ripulita dal rumore e dalle imperfezioni e infine trasformata in una immagine B/N, formata da zero e uno. A partire da questa immagine si calcola l'immagine direzionale dell'impronta, dividendo l'immagine in tante piccole celle immaginarie (come se fosse una matrice) e calcolando la tangente ad ogni cresta presente nelle varie celle, ossia l'angolo di inclinazione. Questo metodo è molto robusto perché è invariante rispetto alle rotazioni dell'impronta e permette quindi un corretto riconoscimento anche quando il dito viene poggiato sul sensore con angolazioni differenti. Una volta ottenuta l'immagine direzionale si procede al rilevamento delle minuzie usando algoritmi molto complicati, molti dei quali sono protetti dal segreto più assoluto.

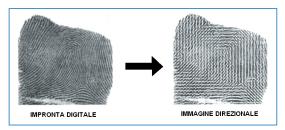



Fig. 3: IMMAGINE DIREZIONALE
L'immagine direzionale di una impronta è la rappresentazione elettronica dell'impronta stessa. Si calcola dividendo l'immagine una sorta di matrice a blocchi e calcolando la tangente ad ogni cresta (l'angolo di inclinazione) presente nelle varie celle.

MAGICSECURE SDK

Il sistema biometrico MagicSecure progettato dalla Eutron si basa su un sensore di rilevamento impronte (venduto in versione singola o integrato su un mouse ottico) dotato di un kit di autenticazione per sistemi Windows 2000 e XP. Il software comprende il driver USB necessario per pilotare il sensore e un rimpiazzamento del modulo *MSGINA.DLL di Windows* (di cui si è a lungo parlato nell'articolo precedente) che consente agli utenti di autenticarsi usando la normale password o ricorrendo all'impronta digitale.
L'installazione del kit prevede inoltre una utility di migrazione account utile per convertire le informazioni e gli account esistenti sul nuovo modulo GINA, con la possibilità di memorizzare fino a 10 dita per ciascun utente. Ma il motivo per cui ioProgrammo ha scelto di trattare MagicSecure in questo articolo è un altro: gli sviluppatori interessati all'argomento possono infatti acquistare (separatamente) l'SDK con le librerie, la documentazione (in lingua inglese) e gli esempi necessari per sviluppare software di autenticazione e riconoscimento mediante impronte digitali, in diversi linguaggi (C++, Visual Basic, Delphi). L'SDK è basato interamente sulla libreria HBAPI.DLL che esporta una API utilizzabile per pilotare il sensore biometrico e per rilevare, visualizzare e confrontare le impronte digitali. La API è il risultato degli sforzi dell'azienda internazionale *Hunno Technologies Inc.* (*www.hunno.com*), specializzata da anni in tecnologie e sensori biometrici. La libreria è divisa in due moduli: *High-Level API* e *Low-Level API*, cioè i programmatori, a seconda della propria competenza e delle proprie necessità, possono importare funzioni con difficoltà implementativa diversa, con la particolarità che le funzioni High-Level sono completamente automatizzate (wizard) nelle procedure di acquisizione e riconoscimento impronte.

LA NOSTRA APPLICAZIONE BIOMETRICA

L'applicazione di esempio si basa sostanzialmente su due operazioni : acquisizione e riconoscimento, implementate come routine di due diversi pulsanti di un form Visual Basic. In aggiunta sul form troviamo alcuni controlli *OptionButton* usati per consentire all'utente di selezionare il livello di soglia del riconoscimento, che è memorizzato nella variabile globale *m_nSecuLevel* (vedi definizioni di FRR e FER). All'avvio il programma cercherà di aprire e

Fig. 4: PASSWORD O IMPRONTA?
Il kit MagicSecure venduto da Eutron consente agli utenti di autenticarsi e accedere a Windows in 4 diverse modalità, usando la tradizionale password in combinazione con le impronte digitali.

Biometria

Password
a "portata di mano"

Prestazioni

La misura delle prestazioni di un sistema biometrico si basa sui seguenti indici:

- ***FRR (False Rejection Rate):*** **misura la frequenza con cui il sistema rifiuta gli utenti legittimi;**
- ***FAR (False Acceptance Rate):*** **misura la frequenza con cui il sistema consente l'accesso ad utenti non autorizzati;**
- ***ERR (Equal Error Rate):*** **errore del sistema nel punto in cui vale che *FRR = FAR;***

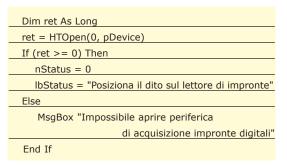
In genere *FRR(k)* e *FAR(k)* sono due funzioni del parametro *"k"*, detto *valore di soglia del sistema*. Il variare di *k* porta in genere all'aumento di un indice e al conseguente decremento dell'altro.

Biometria

Password
a "portata di mano"

Fig. 5: DAMMI LA MANO E TI DIRO' CHI SEI L'applicazione dimostrativa creata con l'SDK consente di acquisire le impronte di più utenti per poi salvarle su file, creando un archivio personalizzato. In seguito, usando il pulsante Riconosci, sarà possibile rilevare l'identità di una persona schedata in archivio.

leggere il file *"C:\USERS.DAT"* (indispensabile!), usato come contatore per memorizzare il numero di utenti correntemente schedati dal sistema di riconoscimento. Il progetto VB richiede inoltre, per un corretto funzionamento, la presenza dell'SDK e deve contenere un modulo aggiuntivo con le dichiarazioni delle funzioni, dei tipi e delle costanti importate da *HBAPI.DLL*. Nel nostro progetto abbiamo fatto uso della Low-Level API per l'acquisizione e della High-Level API per il riconoscimento. L'apertura della periferica di riconoscimento USB e il test della sua presenza avvengono così:

```
Dim ret As Long
ret = HTOpen(0, pDevice)
If (ret >= 0) Then
    nStatus = 0
    lbStatus = "Posiziona il dito sul lettore di impronte"
Else
    MsgBox "Impossibile aprire periferica
                di acquisizione impronte digitali"
End If
```

Il codice è auto-esplicativo: si fa uso della funzione *HTOpen* che restituisce un valore *Long* maggiore o uguale a zero se l'operazione è riuscita; i parametri da passare alla funzione sono l'ID della periferica e una variabile globale (*pDevice*) in cui verrà memorizzato il codice associato alla periferica, da usare in tutte le successive operazioni. Ovviamente in fase di uscita bisognerà chiudere e rilasciare il controllo del lettore di impronte usando *HTClose*. L'acquisizione di impronte ad alto livello è immediata, basta solo chiamare *HTEnroll* passando i parametri necessari e il gioco è fatto: questa chiamata genera infatti una finestra indipendente, dotata di icone e grafica, in cui è possibile acquisire l'impronta di un qualsiasi dito in modo molto semplice; al programmatore non è richiesto nessuno sforzo. L'acquisizione a basso livello è invece più macchinosa e richiede l'uso di un timer col quale temporizzare le diverse fasi dell'acquisizione (periferica pronta, contatto del dito sul sensore, rilascio del dito):

```
Dim pulReadStatus As Long
 Dim Result As Long
 Dim Buffer() As Byte
 'Lettura singola impronta digitale usando Low-Level API
 'Parametri da passare:
 '  periferica da usare (ricavata da HTOpen)
 '  variabile di tipo HT_RAW_DATA che conterrà
                        l'immagine (raw) dell'impronta letta
 '  variabile di tipo HT_BIO_DATA (dati biometrici
                        calcolati dall'impronta)
 '  stato della periferica
 'Restituisce :
 '  boolean (true se l'operazione riesce)
 Dim pRawData As HT_RAW_DATA
 Dim pEnRollBioData As HT_BIO_DATA
 Result = HTRead(pDevice, pRawData,
                        pEnRollBioData, pulReadStatus)
 If Result = HT_OK Then
    'Disegna su un frame immagine VB l'impronta
                        catturata in pRawData
    Result = HTDrawRawImage(Frame.hWnd, 8, 12,
                        160, 160, pRawData)
```

Fig. 6: LETTURA DELL'IMPRONTA Come un moderno mago, il sensore biometrico legge la nostra mano, visualizzando le impronte sul form....sarà anche in grado di prevedere il nostro futuro?

L'impronta digitale, una volta acquisita viene restituita dalla funzione *HTRead* in due formati diversi: come *HT_RAW_DATA* (corrisponde all'immagine vera e propria acquisita dal sensore) e come *HT_BIO_DATA* (è la traduzione dell'impronta in forma biometrica, analizzabile dal computer). La memorizzazione su file richiede l'intervento di una piccola conversione (eseguita con la API di sistema *CopyMemory*), necessaria per consentire a Visual Basic di trattare adeguatamente i dati biometrici:

```
'Converte i dati biometrici acquisiti dal lettore
'in un array di bytes (necessario per l'uso sotto VB)
ReDim Buffer(pEnRollBioData.ulLength - 1)
CopyMemory Buffer(0), ByVal pEnRollBioData.pvData,
                        pEnRollBioData.ulLength
'Richiede il nome da associare all'impronta acquisita
'e lo scrive su file
nome = InputBox("Nome?", "Assegnazione impronta")
```

Sul Web

Consorzio BioAPI, fondato nel 1998, si propone l'obiettivo di distribuire uno standard per la progettazione di applicazioni biometriche "OS indipendent."
http://www.bioapi.org

Biometric Systems Lab dell'Università di Bologna; ottimo repository di informazioni e pubblicazioni sulle tecnologie biometriche.
http://bias.csr.unibo.it/research/biolab/bio_tree.html

```
Open "C:\FINGER" + FP_RecTest.ID + ".TXT"
                                    For Output As #1
Print #1, nome
Close #1
'Scrive su file i dati biometrici dell'impronta acquisita
Open "C:\FINGER" + FP_RecTest.ID + ".DAT"
        For Binary As #1 Len = pEnRollBioData.ulLength
Put #1, , Buffer
Close #1
```

L'impronta di ciascun utente viene infine scritta su un file di nome *"C:\FINGERxx.DAT"* dove *"xx"* rappresenta il valore numerico del contatore salvato in *"C:\USERS.DAT"*, che viene incrementato ad ogni acquisizione. Ad ogni file che memorizza un'impronta è associato un file analogo del tipo *"C:\FINGERxx.TXT"* che memorizza invece il nome associato alla persona (è un'alternativa scelta per evitare l'uso di un database).

Fig. 7: CONOSCERE... E RICONOSCERE
Il riconoscimento di un'impronta è immediato (si tratta di pochi secondi) e la percentuale di errore è molto bassa. L'applicazione consente tuttavia all'utente di scegliere un adeguato livello di sicurezza, per evitare riconoscimenti errati o troppo serveri.

In fase di riconoscimento bisognerà invece leggere una nuova impronta dal sensore (usando *HTCapture*) e confrontarla poi con tutte quelle memorizzate nei diversi file *"C:\FINGERxx.DAT"*.

```
Dim pBioData As HT_BIO_DATA
Result = HTCapture(FP_RecTest.hWnd, pBioData, 10,
                                          0, 0, 0)
If Result = HT_OK Then
  For I = 1 To n
       Open "C:\FINGER" + CStr(I) + ".DAT" For
                            Binary As #1 Len = 256
      Get #1, , Buffer
      Close #1
      Dim fileBioData1 As HT_BIO_DATA
      fileBioData1.pvData = VarPtr(Buffer(0))
      fileBioData1.ulLength = 256
       'Confronta due impronte digitali usando
                                    High-Level API
      'Parametri da passare:
      '  impronta1 (tipo HT_BIO_DATA),
      '  impronta2 (tipo HT_BIO_DATA),
      '  livello di sicurezza da usare per il confronto
      '  variabile long che indica l'esito del confronto
      'Restituisce :
      '  boolean (true se l'operazione riesce)
       Result = HTMatch(fileBioData1, pBioData,
                          m_nSecuLevel, lRtnMatch_Frt)
```

La funzione *HTMatch* è il cuore dell'SDK, perché implementa il confronto fra due impronte passate come argomento (*fileBioData1* e *pBioData*), usando un certo valore di soglia (*m_nSecuLevel*) e restituendo come esito del confronto *lRtnMatch_Frt*, che vale *HT_SUCCEEDED* se le due impronte combaciano.

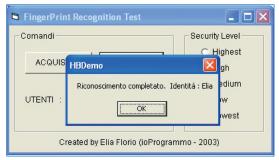



Fig. 8: PIU' PRECISO DI SHERLOCK HOLMES
Da un'impronta riconosciuta diventa facile risalire all'identità di una persona usando un database o un archivio che associa i nomi e i dati anagrafici alle impronte digitali acquisite.

Per il codice completo del programma si rimanda al progetto VB incluso nel CD-ROM della rivista.

Elia Florio

EUTRON

La Eutron è una società da anni leader nel settore dell'IT Security, produttrice di diverse soluzioni per la protezione del software, del copyright e per l'autenticazione dell'identità mediante token, chiavi hardware e ora anche attraverso le impronte digitali. Tutte le informazioni sul kit MagicSecure distribuito dalla Eutron sono reperibili all'indirizzo http://www.eutron.it/infosecurity/magic.asp

il costo dei diversi kit biometrici venduti da Eutron varia tra i 130 e i 180 euro (IVA esclusa) mentre l'SDK necessario per lo sviluppo delle applicazioni viene venduto separatamente.

Biometria

Password
a "portata di mano"

MagicSecure SDK

Il sistema biometrico *MagicSecure* progettato dalla Eutron si basa su un sensore di rilevamento impronte (venduto in versione singola o integrato su un mouse ottico) dotato di un kit di autenticazione per sistemi Windows 2000 e XP. Il software comprende il driver USB necessario per pilotare il sensore e un rimpiazzamento del modulo *MSGINA.DLL di Windows*.

Librerie

DI COMPRESSIONE OPEN SOURCE

Sistema

In questo articolo vedremo come introdurre nelle nostre applicazioni, in modo semplice e rapido, la possibilità di gestire dati in forma compressa, sia su disco che direttamente in memoria. L'utilizzo di alcune delle tecniche illustrate, oltre a ridurre la quantità di spazio necessario a memorizzare i nostri dati, fornisce una prima forma di cifratura di informazioni che non devono essere facilmente rilevabili analizzando i file che fanno parte delle nostre applicazioni.

File sul CD
\soft\codice\compressione\

Lossy
Indica un tipo di compressione che a fronte di una perdita parziale di informazioni (es. formato JPEG) permette di ottenere livelli di compressione più elevati di quelli ottenibili con algoritmi di tipo lossless.

Chi ha sviluppato progetti di una certa entità si sarà probabilmente imbattuto nel problema di dover riorganizzare i propri database, immagini o dati in genere, che per ragioni di dimensioni sforano il tetto massimo che ci si è prefissati o che le specifiche impongono. Cosa fare, ad esempio, se le textures del nostro ultimo gioco 3D sono troppe o troppo dettagliate ma non vi vogliamo o non vi possiamo rinunciare? Il primo pensiero corre a famosi programmi di archiviazione come WinZIP, WinRAR e simili. Certo una soluzione, ma come gestire il tutto direttamente dal nostro codice? Basta dare un'occhita in giro per la rete per rendersi conto che in realtà esistono ottime soluzioni al problema e per di più totalmente gratuite: librerie di compressione lossless più o meno efficienti e con API più o meno semplici da usare. Le librerie di compressione proposte di seguito sono state scritte nella maggior parte dei casi in puro ANSI C, e dunque facilmente portabili su qualunque sistema operativo che fornisca un compilatore C che rispetti tali specifiche. Per loro natura, quindi, tali librerie sono da utilizzarsi principalmente in C, anche se esistono, in alcuni casi, librerie DLL per Windows e quindi utilizzabili, in linea di principio, da qualunque linguaggio/ambiente di programmazione (Delphi, Visual Basic, ecc.). Come vedremo tra breve con esempi pratici di utilizzo, le librerie presentano una interfaccia utente per la compressione/decompressione dei dati secondo due diverse modalità:

- Direttamente in memoria
- Files su disco

La prima delle due possibilità è certamente quella più interessante perché ad esempio ci permette di effettuare una fase di precompressione "off-line" dei dati delle applicazioni, magari usando livelli di compressione molto elevati, e utilizzare i files così ottenuti decomprimendoli "on the fly" in memoria prima di darli in pasto alla parte di codice che dovrà elaborarli. Tale tecnica, se applicata correttamente, porta dei grossi vantaggi per la ridistribuzione delle proprie applicazioni, al costo di un ragionevole impatto sulle prestazioni. La compressione dei dati presenta inoltre altri vantaggi e talvolta inaspettati effetti collaterali:

- Aumento dello spazio disponibile su floppy, ZIP, cdrom, nastri, hard disk e qualunque altro supporto di memorizzazione.
- Una certa probabilità di correzione dei possibili errori generati dai supporti di memorizzazione.
- Riduzione dei tempi di trasferimento in applicazioni di rete e conseguente ottimizzazione dell'utilizzo della banda.
- Un primo semplice livello di protezione dei dati stessi: i files e i flussi di dati in memoria saranno, in senso lato, cifrati.

Nel seguito analizzeremo le API fornite da tre librerie di compressione dati (LibBzip2, LZO e Zlib) indicando vantaggi e svantaggi di ciascuna, e fornendo alcuni esempi di utilizzo delle loro primitive. I sistemi operativi presi come riferimento, sia per la compilazione delle distribuzioni delle stesse librerie che per gli esempi proposti, saranno quelli Unix-like.

LIBBZIP2: EFFICIENTE MA NON VELOCISSIMA

Questa libreria si basa sull'algoritmo di compres-

Sistema

Librerie *di compressione Open Source*

sione di Burrows-Wheeler, che mediamente si comporta meglio della più famosa famiglia di algoritmi LZ77/LZ78 anche se non eccelle in quanto a velocità di compressione. Questa libreria è disponibile per una vasta gamma di architetture, tra le quali Linux, Windows 95/98/ME/NT 4/2000/XP (in versione statica e DLL), MacOS, Solaris Sparc e x86, OS/2, OpenBSD e FreeBSD.
L'interfaccia utente fornita da LibBzip2 si presenta suddivisa su tre livelli: funzioni di basso livello, funzioni di alto livello e funzioni di utilità. Al livello più basso troviamo le funzioni per comprimere/decomprimere direttamente in memoria:

- **int BZ2_bzCompressInit**(*bz_stream *stream, int blockSize, int verbosity, int workFactor)*
- **int BZ2_bzCompress**(*bz_stream *stream, int action)*
- **int BZ2_bzCompresEnd**(*bz_stream *stream)*
- **int BZ2_bzDecompressInit**(*bz_stream *stream, int verbosity, int small)*
- **int BZ2_bzDecompress**(*bz_stream *stream)*
- **int BZ2_bzDecompressEnd**(*bz_stream *stream)*

Definizione del tipo bz_stream in bzlib.h

```
typedef  struct {
      char *next_in;
      unsigned int avail_in;
      unsigned int total_in_lo32;
      unsigned int total_in_hi32;
      char *next_out;
      unsigned int avail_out;
      unsigned int total_out_lo32;
      unsigned int total_out_hi32;
      void *state;
      void *(*bzalloc)(void *, int, int);
      void (*bzfree)(void *,void *);
      void *opaque;
   }
   bz_stream;
```

Un tipo *bz_stream*, come è possibile vedere, indica una struttura in cui sono mantenute informazioni sui buffer sorgente e destinazione utilizzati durante la compressione/decompressione, *blockSize= [1, … ,9]* è la dimensione del blocco dati usato per la compressione (9=massima compressione ed elevato uso della memoria), *verbosity=[0,…,4]* indica il livello di verbosità da usare (0=nessun messaggio), mentre *workFactor=[0,..,250]* indica la soglia oltre la quale si passa dall'algoritmo standard a quello di tipo *fallback* in presenza di blocchi di dati ripetitivi, computazionalmente piuttosto pesanti da comprimere. Per quanto riguarda il parametro *small*, se è settato ad un valore diverso da zero forza al libreria ad usare un algoritmo di decompressione alternativo che riduce l'utilizzo della memoria. Sebbene il tutto possa sembrare complicato, in realtà la sequenza di operazioni da compiere per comprimere un blocco di dati è piuttosto semplice:

1) Invocare *BZ2_bzCompressInit(&stream, 4, 0, 30)* specificando i puntatori ai buffer sorgente/destinazione nella struttura puntata da *&stream*
2) Invocare *BZ2_bzCompress(&stream, BZ_RUN)*
3) Modificare opportunamente i puntatori ai buffer sorgente/destinazione e invocare *BZ2_bzCompress(&stream, BZ_FINISH)* finché ci sono dati da comprimere
4) Invocare *BZ2_bzCompresEnd(&stream)*

Se il blocco dati da comprimere non è molto grande si può invocare una sola volta la *BZ2_bzCompress()* specificando direttamente il valore *BZ_FINISH* per il parametro *action*. Allo stesso modo, per decomprimere un blocco di dati in memoria basta:

1) Invocare *BZ2_bzDecompressInit(&stream, 0, 0)* specificando i puntatori ai buffer sorgente/destinazione
2) Invocare ripetutamente *BZ2_bzDecompress(&stream)* aggiustando di volta in volta i puntatori ai buffer sorgente/destinazione
3) Invocare *BZ2_bzDecompressEnd(&stream)* per concludere l'operazione

Vediamo un breve esempio di come sia possibile comprimere e decomprimere un blocco di dati in memoria usando queste primitive (esempio presente sul CD: *ESEMPIO 1: bzip2_LowLevel.c)*.
L'interfaccia di alto livello fornisce una serie di primitive che permettono di comprimere e decomprimere direttamente i file, che per default sono identificati dal suffisso *.bz2*:

- **BZFILE *BZ2_bzReadOpen**(*int *bzerror, FILE *fp, int small, int verbosity)*
- **int BZ2_bzRead**(*int *bzerror, BZFILE *bzfp, void *buf, int len)*
- **void BZ2_bzReadGetUnused**(*int *bzerror, BZFILE *bzfp, void **unused, int *nUnused)*
- **void BZ2_bzReadClose**(*int *bzerror, BZFILE*

Burrows-Wheeler

La trasformata di Burrows-Wheeler è una operazione che permette di ottenere, a partire da una stringa *S* di *N* caratteri, una matrice *M* le cui righe sono tutti gli shift ciclici a sinistra della stringa *S*. L'ultima colonna *BW* della matrice *M* ordinata costituisce la trasformata di Burrows-Wheeler. La stringa *BW* così ottenuta contiene gli stessi caratteri della stringa in input, ma è più facile da comprimere.

bzfp)

- **BZFILE *BZ2_bzWriteOpen***(int *bzerror, FILE *fp, int blockSize, int verbosity, int workFactor)*
- **void BZ2_bzWrite***(int *bzerror, BZFILE *bzfp, void *buf, int len)*
- **void BZ2_bzWriteClose***(int *bzerror, BZFILE *bzfp, int abandon, unsigned int *nbytes_in, unsigned int *nbytes_out)*

Il prossimo esempio chiarisce come utilizzare tali primitive (esempio presente sul CD: *ESEMPIO 2: bzip2_ HiLevel.c).* Le funzioni di utilità, infine, permettono di comprimere e decomprimere i dati in memoria attraverso una semplice chiamata alle seguenti funzioni:

- **int BZ2_bzBuffToBuffCompress***(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int blockSize, int verbosity, int workFactor)*
- **int BZ2_bzBuffToBuffDecompress***(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int small, int verbosity)*

i cui parametri hanno la stessa semantica di quelli visti in precedenza per le funzioni di alto livello (esempio presente sul CD: *ESEMPIO 3: bzip2_Utility.c)*. Per compilare gli esempi precedenti, basta seguire la solita procedura:

- Scaricare il tarball *bzip2-1.0.2.tar.gz* dal sito *sources.redhat.com/bzip2*
- Compilare il tutto col comando "make" per ottenere la libreria *libbz2.a*
- Compilare gli esempi da riga di comando: *cc <sorgente.c> -lbz2*

L'unico svantaggio di questa libreria risiede nell'affermazione fatta dagli autori della stessa, e suffragata dall'evidenza dei benchmark, ossia nella non trascurabile quantità di memoria e cicli di CPU richiesti per la compressione e decompressione dei dati. Per questo motivo LibBzip2 non è la scelta ottimale nel caso in cui l'interattività risulta essere un fattore chiave dell'applicazione. Ma questo probabilmente è da imputare alla complessità computazionale della trasformata di Burrows-Wheeler.

LZ77/LZ78

Sono due varianti di una serie di algoritmi detti di Lempel-Ziv. Concettualmente l'algoritmo LZ78 è una variante di LZ77 che consente di raggiungere una compressione di 4:1. Invece di caricare i dati in un buffer e sostituire i byte ripetuti solamente entro il blocco attuale, LZ78 crea un elenco dei byte ripetuti e li sostituisce sull'intero file.

LZO (LEMPEL-ZIV-OBERHUMER): DECOMPRESSIONE REAL-TIME

Questa libreria di compressione è considerata come una delle più efficienti in termini di tempo necessario a comprimere/decomprimere i dati: la velocità permette di operare la decompressione, e spesso anche la compressione, in real-time. Inoltre è molto efficiente in termini di memoria richiesta per le operazioni: sia la compressione che la decompressione possono essere di tipo overlapped, ossia è possibile comprimere o decomprimere i dati nella stessa area in cui questi sono memorizzati, azzerando di fatto la richiesta di memoria extra. Il rovescio della medaglia è che purtroppo non risulta altrettanto efficiente in termini di rapporto di compressione. Per tale motivo, la migliore strategia di utilizzo consiste nel precomprimere i dati da usare nelle applicazioni, e decomprimerli al volo a seconda delle necessità, facendo affidamento sulla capacità di decompressione real-time. In tal modo è possibile usare algoritmi relativamente lenti nella compressione ma molto efficienti.
La libreria LZO comprende molti algoritmi, ciascuno con diversi livelli di compressione: *LZOX-N*, con *X*=tipo di algoritmo *(1, 1A, 1B, 1C, 1F, 1X, 1Y, 1Z, 2A)*, ed *N*=livello di compressione *(1…9, 99, 999)*. Naturalmente, quanto più è elevato il valore di *N* tanto più efficiente e lento sarà il processo di codifica. Nella maggior parte dei casi la scelta migliore risulta essere l'algoritmo *LZO1X-N*.
Anche in questo caso le piattaforme supportate sono le più svariate: Dos (16 e 32 bit), Windows 3.x, Windows 95/98/NT 4/2000/XP, Linux, HPUX, Alpha, AIX, MacOS, e così via. Esistono inoltre versioni native in Java, Perl e Python. L'interfaccia utente in C, linguaggio in cui è scritta la libreria, risulta essere molto estesa, per lasciare al programmatore la più ampia libertà sulla scelta dell'algoritmo da utilizzare, ma di semplice gestione. Rispetto alle altre, tuttavia, non fornisce primitive in grado di creare files con un preciso formato di compressione standard, ma tutto il lavoro è lasciato al programmatore. A tale scopo sono fornite alcune primitive di utilità tra cui quelle per il calcolo del checksum *(Adler* e *CRC)*, *lzo_fwrite()* ed *lzo_fread()* (versioni portabili delle *fread() /fwrite() standard*). Le primitive fornite permettono unicamente di operare la compressione/decompressione di blocchi di dati in memoria. Per operare su uno di questi blocchi basta invocare la funzione *lzo_init()* e quindi passare alla compressione o decompressione dei dati in questione utilizzando le funzioni relative alla versione dell'algoritmo scelto (ad esempio *lzo1x_1_compress()* e *lzo1x_1_decompress()* nel caso si volesse usare *LZO1X-1*). Vediamo in dettaglio queste primitive:

- **int lzo_init*(void)*:** inizializzazione della libreria da invocare prima dell'utilizzo di qualun-

que altra primitiva.

- **int lzo1x_1_compress** ***(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp dst_ len, lzo_voidp wrkmem)*****:** compressione con algoritmo *LZO1X-1*. Il parametro *wrkmem* rappresenta il puntatore ad un buffer utilizzato internamente dalla libreria e la cui dimensione varia in base all'algoritmo usato. La dimensione di questo buffer è predefinita da una serie di costanti che ritroviamo nell'header file relativo all'algoritmo. Per quanto riguarda l'esempio riportato di seguito, nel file *lzo1x.h* è definita la costante *LZO1X_1_ MEM_COMPRESS*.

- **int lzo1x_decompress*****(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp dst_ len, lzo_voidp wrkmem)*****:** decompressione con algoritmo *LZO1X-1*. In questo caso il parametro wrkmem non è utilizzato e va posto al valore *NULL*.

Esiste, come detto, una vasta scelta di primitive: in genere per ciascun algoritmo esiste una serie di funzioni del tipo *lzoALG_N_compress()* ed *lzoALG_ N_decompressTYPE()*, dove:

- *ALG=1, 1a, 1b, 1c, 1f, 1x, 1y, 1z, 2a*

- *N=1,2,3,4,5,6,7,8,9,99,999*

- *TYPE= _safe, _asm, _asm_safe, _asm_fast, _asm_fast_safe*. Il suffisso *_asm* indica il fatto che il codice è per lo più scritto in assembly ottimizzato, *_safe* indica la versione di decompressore che effettua alcuni test di validità del blocco dati compresso da elaborare, mentre *_fast* indica una versione ancora più performante in termini di velocità di esecuzione.

Per quello che concerne il buffer in cui i dati verranno decompressi, esistono semplici formule per calcolarne la dimensione:

- Per gli algoritmi *LZO1* out_buffer_size = in_buffer_size+(in_buffer_size/64)+16+3.*

- Per gli algoritmi *LZO2* out_buffer_size = in_buffer_size+(in_buffer_size/8)+128+3.*

La libreria *LZO* può essere scaricata da *www.oberhumer.com/opensource/lzo* e compilata con la seguente procedura:

- *configure*
- *make*
- *make install*

A questo punto è possibile compilare l'esempio che usa le funzioni relative all'algoritmo LZO1X-1 con un semplice comando: (esempio presente sul CD: *ESEMPIO 4: lzo_compress.c).*

```
cc –o lzo.compress lzo.compress.c –llzo
```

Oltre alla versione completa, esiste anche una versione minimale della libreria LZO, chiamata *miniLZO*, costituita da un sorgente in C e due header files, che occupa solo 6KB dopo la compilazione, mentre l'intera distribuzione occupa appena 14 KB. Come dire: nessuna scusa per non incorporare queste funzionalità nella proprie applicazioni.

ZLIB: UN BUON COMPROMESSO TRA VELOCITÀ ED EFFICIENZA

Questa libreria utilizza una variante del ben noto algoritmo *LZ77* chiamata deflation (RFC1951), che affonda le sue radici nell'altrettanto famoso PkZip della PkWare.
Anche *Zlib*, come le precedenti, è assolutamente gratuita, scritta in ANSI C e vanta un enorme numero di sistemi operativi su cui è stata implementata: Windows 9x/NT 4/2000 /XP/CE, Linux, MacOS, Solaris, HPUX, Compaq Tru64, Irix 6.x e BeOS, solo per citare i più noti. Il rapporto di compressione può variare da 2:1 a 5:1, anche se, come da specifiche, il formato zlib è capace di un limite teorico di 1030:1.

Definizione del tipo z_streamp in zlib.h.

```
typedef struct z_stream_s {
   Bytef   *next_in; /* next input byte */
   uInt avail_in; /* number of bytes available
                                              at next_in */
   uLong total_in; /* total nb of input bytes read so far */
   Bytef   *next_out; /* next output byte
                                    should be put there */
   uInt    avail_out;/* remaining free space at
                                             next_out */
   uLong total_out;  /* total nb of bytes output so far */
   char *msg; /* last error message, NULL if no error */
   struct internal_state FAR *state; /* not visible by
                                          applications */
   alloc_func zalloc;/* used to allocate the internal
                                               state */
   free_func zfree;  /* used to free the internal state */
   voidpf opaque; /* private data object
                               passed to zalloc and zfree */
   int data_type; /* best guess about the data type:
                                       ascii or binary */
   uLong adler; /* adler32 value of the uncompressed
                                                data */
   uLong reserved; /* reserved for future use */
```

Sistema

Librerie
di compressione Open Source

Adler

Si tratta di una tecnica di calcolo del checksum che impiega 2 contatori da 16 bit, s1 ed *s2*. Il primo contiene la somma di tutti i byte in input, mentre *s2* contiene la somma di tutti i valori di *s1*. Entrambe le somme sono modulo 65521. Il checksum è dato da *s2*65536 + s1* in network byte order.

CRC

Il Cyclic Redundancy Check permette di calcolare il checksum (un valore a 32 bit che riassume il contenuto di un blocco di dati) sommando blocchi da 2 byte in complemento a 1.

Sistema

Librerie
di compressione Open Source

```
} z_stream ;
typedef  z_stream FAR * z_streamp;
```

L'interfaccia verso il programmatore risulta essere composta da tre classi, una di base (*Basic*) che permette un tuning fine di ogni parametro dell'algoritmo, una di alto livello (*Utility*) che presenta funzioni per la compressione/decompressione in memoria e di file in formato standard *gzip (RFC1952)*, ed infine una classe di funzioni avanzate (*Advanced*) che risulta essere abbastanza complessa e di utilizzo piuttosto raro.
Tali classi risultano essere ben strutturate e possono coprire qualunque necessità possa avere il programmatore: ne è una chiara dimostrazione il fatto che sia stata adottata per i loro prodotti da moltissime compagnie leader nel settore del software per PC. Tra queste ricordiamo anzitutto Microsoft, Apple, Adobe, Macromedia, Cisco, Borland e Netscape. Vediamo in dettaglio i prototipi delle funzioni di livello *Utility*:

- **int compress**(*Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen)*
- **int uncompress**(*Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen)*
- **gzFile gzopen**(*const char *path, const char *mode)*
- **int gzread**(*gzFile file, voidp buf, unsigned len)*
- **int gzwrite**(*gzFile file, const voidp buf, unsigned len)*
- **z_off_t gzseek**(*gzFile file, z_off_t offset, int whence)*
- **int gzeof**(*gzFile file)*
- **int gzclose**(*gzFile file)*

I parametri da passare alle funzioni sono piuttosto intuitivi e l'implementazione del codice che effettua la compressione risulta molto semplice, come si può rapidamente dedurre dall'esempio (esempio presente sul CD: *ESEMPIO 5: zlib_Utility.c)*.
Per quanto riguarda la classe *Basic*, troviamo le seguenti funzioni:

- **int deflateInit** *(z_streamp strm, int level)*
- **int deflate** *(z_streamp strm, int flush)*
- **int deflateEnd** *(z_streamp strm)*
- **int inflateInit** *(z_streamp strm)*
- **int inflate** *(z_streamp strm, int flush)*
- **int inflateEnd** *(z_streamp strm)*

Anche in questo caso, i parametri da passare alle funzioni sono concettualmente identici al caso della libreria *LibBzip2*. Anzi, come chiaramente si può vedere confrontando i listati 1 e 2, le strutture che descrivono lo stream di dati da comprimere o decomprimere sono pressoché identiche se si trascurano alcuni piccoli particolari. Si veda l'esempio presente sul CD: *ESEMPIO 6: zlib_Basic.c*).
La compilazione della distribuzione in formato sorgente di Zlib avviene con la solita procedura:

- *configure*
- *make*
- *make install*

mentre per compilare gli esempi forniti basta dare i comandi:

- **cc -o zlib_Utility zlib_Utility.c –lz** per compilare l'esempio relativo alle funzioni della classe *Utility*.
- **cc -o zlib_Basic zlib_Basic.c –lz** per compilare l'esempio relativo alle funzioni della classe *Basic*.

CONCLUSIONI

Come avrete capito, queste librerie di compressione open source permettono di dotare le nostre applicazioni di porzioni di codice, relativamente semplici da implementare, che gestiscono dati compressi con una notevole riduzione di spazio occupato su disco a fronte di un ragionevole overhead in termini di memoria e cicli di CPU.
Molte applicazioni che usiamo tutti i giorni fanno uso di tali librerie e ciò testimonia, più di ogni altra argomentazione, l'indubbia affidabilità e i molti vantaggi che se ne possono trarre.
La caratteristica di essere multipiattaforma, inoltre, permette di avere un impatto praticamente nullo su tutte quelle applicazioni di natura platform-independent.
Insomma, una volta capite le modalità di utilizzo di queste librerie, la possibilità di maneggiare grosse quantità di dati al costo di una frazione delle loro reali dimensioni diventa pressoché trasparente per le nostre applicazioni. Sicuramente un ottimo motivo per aggiungere la compressione la prossima volta che si ha a che fare con quantità titaniche di dati.

Giuseppe Palumbo

Lossless

E' un termine riferito ad una famiglia di algoritmi di compressione che permettono di ridurre la dimensione di qualunque tipo di dati senza perdita di alcuna informazione.

Sul Web

LibBzip2:
http://sources.redhat.com/bzip2

LZO :
http://www.oberhumer.com/opensource/lzo

Zlib :
http://www.gzip.org/zlib

Fotolab

FILTRI FOTOGRAFICI IN VISUAL BASIC

Visual Basic

In questo articolo, suddiviso in due parti, si vogliono presentare alcuni algoritmi di ritocco fotografico finalizzati a migliorare la qualità di immagini bitmap acquisite con lo scanner o con una macchina fotografica digitale e ad estrarne delle caratteristiche, come ad esempio i contorni delle figure.

Un qualche ritocco si rende spesso necessario perché l'acquisizione di immagini comporta inevitabilmente l'introduzione di "rumore elettronico", che si manifesta mediante l'alterazione del colore di qualche pixel, ed anche di qualche distorsione geometrica. Gli algoritmi verranno sviluppati in Visual Basic, partendo da un file in formato BMP, limitandosi ad usare le librerie e i controlli standard di tale linguaggio. Pertanto risulterà agevole la realizzazione degli stessi in altri linguaggi di programmazione.

IL MODELLO RGB

Si ricorda brevemente che le immagini bitmap a colori sono costituite da una griglia di punti, ad esempio 800 x 600 punti, detti pixel, per ciascuno dei quali si memorizza il colore, ottenuto come combinazione dei tre colori primari Red, Green, Blue (Rosso, Verde, Blu). Per ciascuno di questi colori si memorizza il livello di intensità che può variare da 0 = assenza di colore a 255 = massima intensità di colore, e quindi massima luminosità (brightness).

R	G	B	Colore
255	0	0	rosso
0	255	0	verde
255	255	0	giallo
255	255	240	avorio
255	255	255	bianco
128	128	128	grigio
0	0	0	nero

Tab. 1: **Livelli di intensità necessaria a ottenere alcuni colori fondamentali.**

Per memorizzare questo livello di intensità occorrono 8 bit per ciascuno dei colori primari, e quindi in totale 24 bit per pixel. In questo modo si hanno 2^24 = 16.777.216 possibili colori. Alcuni esempi di colori sono indicati nella Tabella 1.
Secondo il modello RGB, quando le intensità dei tre colori primari sono identiche si ottengono dei livelli di grigio; pertanto si possono avere 256 livelli di grigio, graduati dal nero 0, 0, 0 al bianco 255, 255, 255.

IL FORMATO BMP

La scelta di questo formato è stata dettata dalla sua ampia diffusione unitamente alla sua grande semplicità. Innanzitutto si deve sapere che un file BMP è composto di 4 parti:

1. intestazione del file,
2. intestazione dell'immagine,
3. eventuale tavolozza dei colori,
4. i dati sui pixel.

La Tabella 2 rappresenta in dettaglio la struttura di un file BMP, dove in neretto sono evidenziati i campi che vengono utilizzati nel programma *FotoLab* che svilupperemo. Si fa notare che l'immagine viene memorizzata capovolta rispetto a come essa viene visualizzata a video: ovvero le righe di pixel che costituiscono l'immagine vengono memorizzate in ordine inverso, dall'ultima alla prima. Si veda la Fig. 1.

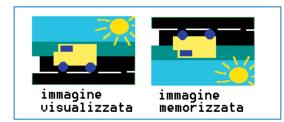



Fig. 1: **Il formato BMP memorizza le immagini capovolte.**

TRASFORMAZIONI DELL'IMMAGINE

Tralasciando tutte le trasformazioni di tipo geometrico (quali ad esempio riflessione, rotazione, ingrandimento, riduzione dell'immagine), ci si occupa delle trasformazioni che modifica-

File sul CD
\soft\codice\filtri.zip

La risoluzione dell'immagine

Il numero di pixel che formano l'immagine acquisita da una fotocamera digitale è fisso, e dipende dalle caratteristiche del suo sensore CCD (Charge Coupled Device): ad esempio essa potrebbe avere un sensore da circa 2 milioni di pixel disposti in una griglia di 1600 x 1200 pixel. Questo vale anche per lo schermo del computer, che viene solitamente impostato per visualizzare immagini a 800x600 pixel, oppure 1024x768 pixel. Tuttavia le dimensioni in pixel non dicono nulla sulle dimensioni fisiche in cm dell'immagine: è necessario considerare la densità dei pixel, ovvero la risoluzione dell'immagine.

Visual Basic

Fotolab
Filtri Fotografici in Visual Basic

no il colore dei pixel, con l'obiettivo di migliorare la qualità di una immagine affetta da rumore, oppure troppo sfuocata, o scura, e di trasformazioni che cercano di evidenziare la superficie oppure i contorni degli oggetti in essa rappresentati. In base alle tecniche utilizzate, le trasformazioni possono essere suddivise in:

- trasformazioni puntuali, che agiscono su ciascun pixel singolarmente;
- trasformazioni locali che modificano il colore di un pixel in funzione dei pixel che gli stanno attorno,
- trasformazioni globali che modificano il colore di un pixel in funzione di tutti i pixel dell'immagine: si tratta ad esempio della trasformata di Fourier per l'analisi delle frequenze spaziali dell'immagine (essa non viene considerata in questo articolo).

TRASFORMAZIONI PUNTUALI

Dato il generico pixel, indichiamo con x(R),

Nome Campo	Dimensione	Descrizione
Intestazione del File	totale 14 byte	
Firma	2 byte = String*2	Contiene i caratteri 'BM'
Dimensione File	4 byte = Long	Dimensione del file in byte
riservato	4 byte	non utilizzato (= 0)
Offset	4 byte = Long	Spiazzamento dei dati relativi ai pixel
Intestazione dell'Immagine	totale 40 byte	
Dimensione dell'intestazione	4 byte = Long	Dimensione dell'intestazione in byte = 40
W	4 byte = Long	Larghezza Immagine
H	4 byte = Long	Altezza Immagine
Numero Piani	2 byte = Integer	Numero di piani del disegno (= 1)
Bit per Pixel	2 byte = Integer	Bit per Pixel 1 = monocromatica. Numero Colori = 1 4 = 4 bit. Numero Colori = 16 8 = 8 bit. Numero Colori = 256 16 = 16 bit. Numero Colori = 65.536 24 = 24 bit RGB. Numero Colori = 16 milioni
Compressione	4 byte = Long	Tipo di Compressione 0 = RGB nessuna compressione 1 = RLE8 codifica RLE a 8 bit 2 = RLE4 codifica RLE a 4 bit
Dimensione dell'Immagine Compressa	4 byte = Long	= 0 se Tipo di Compressione = 0
Pixel Per Metro Asse X	4 byte = Long	Scala orizzontale Pixel/metro
Pixel Per Metro Asse Y	4 byte = Long	Scala verticale Pixel/metro
Numero Colori 4 Usati	byte = Long	Numero di colori effettivamente usati
Colori Importanti	4 byte = Long	Numero di colori importanti = 0 se tutti
Tavolozza dei colori	totale (4 * Num. Colori Usati) byte	È presente solo se Bit Per Pixel <= 8
Blue	1 byte	Intensità di Blu
Green	1 byte	Intensità di Verde
Red	1 byte	Intensità di Rosso
riservato	1 byte	Non utilizzato (= 0)
... ripetuto per tutti i colori usati		
Dati sui Pixel	totale byte = multiplo4(3 * W) * H	Il totale di byte per riga = 3 * W deve essere arrotondato per eccesso ad un multiplo di 4: se ad esempio W = 5 pixel e H = 5 pixel, allora 3 * W = 15 e quindi multiplo4(3 * W) = 16, pertanto il totale sarà 16 * 5 = 80 byte. I byte aggiunti per arrotondare sono settati a 0
Blue	1 byte	Intensità di Blu
Green	1 byte	Intensità di Verde
Red	1 byte	Intensità di Rosso
... ripetuto per tutti i pixel dell'immagine		

Tab. 2: **Dettaglio della struttura di un file BMP.**

La dimensione dell'immagine

La risoluzione di una immagine è il rapporto tra il numero dei pixel che la costituiscono e la dimensione fisica della stessa; essa si misura in dpi (dot per inch = punti per pollice). Si ricorda l'equivalenza 1 pollice = 2.54 cm. Un normale monitor per computer ha una risoluzione relativamente bassa, di 72 dpi, mentre una stampante a getto d'inchiostro può avere una risoluzione di 600 dpi. Uno scanner piano per fogli A4 può avere una risoluzione effettiva di 2400 dpi, uno scanner per pellicole una risoluzione di 2700 dpi.
La stessa immagine può essere riprodotta su carta con risoluzioni e quindi dimensioni diverse; ad esempio una foto di 1268 x 1012 pixel se stampata ad una risoluzione di 300 dpi avrà dimensioni di 4.2 x 3.4 pollici, ovvero di 10.7 x 8.6 cm (infatti 1268 / 300 = 4.2 e 1012 / 300 = 8.6); se invece la si vuole stampare con dimensioni raddoppiate su entrambi i lati, la risoluzione si dimezza a 150 dpi.

x(G), x(B) le intensità, rispettivamente, delle sue componenti rossa, verde e blu. Queste intensità costituiscono le tre componenti di colore di ciascun pixel. Le trasformazioni puntuali consentono di ottenere delle nuove intensità y(R), y(G) e y(B) in funzione soltanto delle corrispondenti componenti x(R), x(G), x(B):

1) La trasformazione che produce il negativo di una immagine consiste nel calcolare il complemento a 255 dell'intensità di ciascuno dei tre colori di base di tutti i pixel, secondo la formula

   $y = 255 - x$

   La Fig. 2 evidenzia graficamente l'andamento lineare di questa trasformazione.

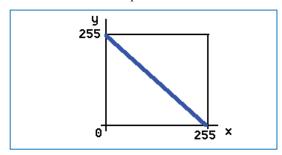



Fig. 2: **Inversione di una immagine.**

2) Una trasformazione che consente di aggiustare in modo semplice la luminosità e il contrasto di una immagine è la trasformazione lineare delle intensità di colore

   Per aumentare la luminosità si deve aumentare il livello di intensità dei colori dei pixel, mentre per aumentare il contrasto si deve aumentare la differenza tra le intensità dei colori di pixel adiacenti.
   La formula è la seguente

   $y = a * x + b$

   dove i parametri a e b sono numeri reali che rappresentano, rispettivamente, il fattore di contrasto e la variazione di luminosità. Il contrasto aumenta per a > 1 e diminuisce per 0 < a < 1, mentre la luminosità aumenta per b > 0 e diminuisce per b < 0. I valori di y ottenuti da tale calcolo vengono ricondotti all'intervallo 0 .. 255, assegnando il valore 0 agli y negativi e il valore 255 agli y maggiori di 255. La Fig. 3 illustra il significato grafico dei parametri a e b, i quali rappresentano, rispettivamente, la pendenza della retta e la sua staccata rispetto all'origine.

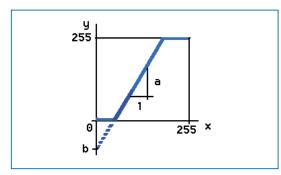



Fig. 3: **Trasformazione lineare**

3) Quando i tre colori di base hanno uguale intensità, si ha una immagine a 256 livelli di grigio.

Per separare nettamente le zone chiare dell'immagine da quelle scure si può applicare la trasformazione di sogliatura (thresholding) che riconduce l'immagine a due soli livelli di grigio: nero e bianco. Lo scopo di questa trasformazione è quello di segmentare l'immagine, ovvero di evidenziare la superficie degli oggetti raffigurati nella stessa. Essa risulta maggiormente efficace nei casi in cui si ha un oggetto di colore omogeneo che campeggia su uno sfondo anch'esso di colore omogeneo. A causa del rumore elettronico non ci sarà mai una perfetta distinzione tra l'oggetto e lo sfondo e quindi questa operazione sarà comunque sempre soggetta ad un qualche errore. La tecnica della trasformazione di sogliatura è molto semplice: dato un valore di soglia *s*, compreso tra 0 e 255, si fanno diventare neri tutti i pixel il cui livello di grigio è inferiore a *s* e bianchi tutti gli altri:

$$\begin{cases} y = 0 & \text{per } x < s \\ y = 255 & \text{per } x >= s \end{cases}$$

La Fig. 4 evidenzia graficamente tale comportamento.

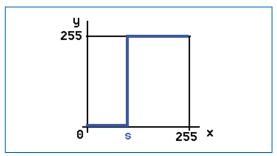



Fig. 4: **Sogliatura.**

La scelta del valore di soglia s si basa sull'analisi dell'istogramma delle frequenze dei livelli di grigio dell'immagine.
La Fig. 5 riporta un esempio di tale istogramma; esso evidenzia il fatto che nell'immagine ci

L'istogramma dei livelli di grigio

L'istogramma di una immagine a livelli di grigio è un strumento utile per caratterizzare l'immagine stessa. Esso rappresenta, per ognuno dei 256 livelli di grigio, il numero di pixel dell'immagine che assumono tale livello; tali valori verranno poi divisi per il numero totale di pixel dell'immagine in modo da ottenere il diagramma delle frequenze relative dei livelli di grigio.

Visual Basic

Fotolab

Filtri Fotografici in Visual Basic

Fig. 5: Istogramma dei livelli di grigio.

sono due distinte aree: una più scura, di intensità compresa tra 30 e 70 e l'altra più chiara, di intensità compresa tra *110* e *150*. Pertanto, in questo caso, la scelta del valore di soglia *s = 100* consente di separare nettamente le due zone dell'immagine, facendo diventare neri tutti i pixel con intensità < *100* e bianchi tutti gli altri. Ovviamente qualsiasi valore di soglia venga scelto compreso nell'intervallo tra *70* e *110* darà il medesimo risultato di *s = 100*.
L'esempio fornito illustra una situazione particolarmente favorevole, ma spesso non c'è una così netta distinzione tra le intensità di colore delle diverse aree dell'immagine e pertanto la scelta del valore di soglia pur se fatta con oculatezza non consentirà una netta evidenziazione delle stesse.

La tecnica dei falsi colori

Si tratta di una tecnica che consente la segmentazione dell'immagine, ovvero di evidenziare alcune zone della stessa, caratterizzate da una omogeneità cromatica.
Data una immagine a livelli di grigio, si fissano degli intervalli di intensità di grigio e si associa arbitrariamente a ciascuno di essi un diverso colore: si tratta evidentemente di "falsi colori".
Essi sono finalizzati a facilitare la lettura dell'immagine, anche perchè l'occhio umano non è in grado di distinguere tutti i 256 livelli di grigio, ma soltanto un sessantina.
Per scegliere tali intervalli si farà uso dell'istogramma dei livelli di grigio.
Ad esempio: i pixel con intensità comprese tra 41 e 60 diventeranno verdi, quelli con intensità compresa tra 61 e 80 diventeranno rossi e tutti gli altri manterranno l'attuale intensità di grigio.

ESEMPIO

A titolo esemplificativo si mostrano gli effetti delle suddette elaborazioni sull'immagine di Fig. 6a.

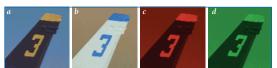



Fig. 6a: Tre; **Fig. 6b:** Negativo; **Fig. 6c:** Rosso; **Fig. 6d:** Verde.

Dell'immagine di partenza ne viene fatto il negativo (Fig. 6b) e ne vengono estratti i colori primari (Fig. 6c, 6d, 6e). Essa viene poi convertita in scala di grigi (Fig. 6f), il cui istogramma dei livelli di grigio è proprio quello rappresentato in Fig. 5, a questo punto viene applicata la preannunciata operazione di sogliatura, appunto con valore di soglia pari a 100 (Fig. 6g).

Fig. 6e: Blu; **Fig. 6f:** Grigio; **Fig. 6g:** Grigio Soglia *100*; **Fig. 6h:** *a = 2 b = 20.*

Ripartendo dalla Fig. 6a si applica una trasformazione lineare per aumentarne il contrasto e la luminosità: la Fig. 6h mostra il risultato applicando un fattore di contrasto pari a 2 e una variazione di luminosità pari a +20.

CODIFICA IN VB

La Fig. 7 mostra la finestra di tipo *"Multiple Document"* del programma *FotoLab* con all'interno due finestre "figlie" contenenti i controlli *PictureBox* per visualizzare le foto coinvolte nelle elaborazioni.

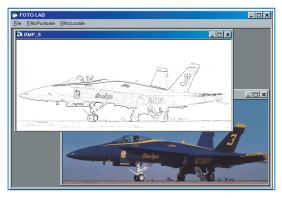



Fig. 7: FotoLab

Il programma applica il filtro prescelto alla foto corrente; per rendere corrente una foto si deve fare un click all'interno della stessa. *FotoLab* gestisce soltanto immagini BMP a 24 bit, ovvero a 16 milioni di colori, e qualora si trasformi una foto a colori in una foto a 256 livelli di grigio, per semplicità, essa viene considerata come un caso particolare di una foto a colori senza adottare il più opportuno formato BMP a 8 bit.
Inoltre, avendo uno scopo didattico, il programma non si preoccupa troppo dell'efficienza degli algoritmi, ne' recupera spazio in memoria centrale quando si chiude la finestra associata ad una foto. A livello globale viene dichiarato un array di oggetti "foto", ciascuno dei quali avrà il nome della foto, il riferimento al Form su cui essa viene visualizzata, le dimensioni della stessa, l'offset dei pixel e gli array di byte contenenti l'intestazione del file e la matrice di pixel.

```
Private Type tipoFoto
  NomeFoto As String
  FormFoto As frmFoto
  w As Long  ' ampiezza foto
  w3 As Long  ' ampiezza aggiustata
  h As Long  ' altezza foto
  inizioPixel As Long  ' offset
  header() As Byte  ' header del file BMP
  g() As Byte  ' matrice di pixel
End Type
Dim Foto() As tipoFoto  ' array di foto
Dim nFoto As Integer  ' numero di foto caricate
                                   nell'array
```

```
Dim gCorrente As Integer  ' indice della foto corrente
```

La seguente subroutine consente di caricare una foto BMP letta da disco in una nuova componente dell'array di foto. Il parametro *NomeFoto* contiene il nome del file e il parametro *Origine* vale 0 per indicare che si deve leggere la foto da disco. La foto ha dimensione *w * h* pixel, quindi essa occuperà una matrice di Byte con *h* righe e *w3 = w * 3* colonne, dove *w3* viene arrotondato al valore multiplo di *4* immediatamente successivo, a causa delle specifiche del formato BMP.

```
Private Sub CaricaFoto(ByVal NomeFoto As String,
                       ByVal Origine as Integer)
Dim w As Long
Dim h As Long
Dim pixelOffset As Long
With Foto(gCorrente)    ' gCorrente è l'indice della foto
                                                 corrente
  Set .FormFoto = New frmFoto  ' crea un nuovo Form
                                              per la foto
  .FormFoto.Caption = NomeFoto
  .FormFoto.Tag = CStr(gCorrente)  ' associa il form alla
                                            foto corrente
  .FormFoto.Show
  .NomeFoto = NomeFoto
  If Origine = 0 Then
      Set .FormFoto.picFoto.Picture =
                                  LoadPicture(NomeFoto)
    Open NomeFoto For Binary As #1
    Get #1, 11, pixelOffset
    Get #1, 19, w   ' Width
    Get #1, 23, h   ' Height
    .w = w
    .w3 = multiplo4(w * 3) ' funzione che calcola il
                                           multiplo di 4
    .h = h
    .inizioPixel = 1 + pixelOffset
    ' acquisizione header del file BMP
    ReDim .header(1 To pixelOffset)
    Get #1, 1, .header
    ' acquisizione matrice di pixel
    ReDim .g(1 To .w3, 1 To .h)
    ' notare che la matrice risulta trasposta: colonne,
    ' righe e che inoltre le righe sono invertite:
    ' dall'ultima alla prima
    Get #1, .inizioPixel, .g
    Close #1
  Else
    ' la foto viene letta dall'array, dalla componente di
    ' indice Origine
    .w = Foto(Origine).w
    .w3 = Foto(Origine).w3
    .h = Foto(Origine).h
    .inizioPixel = Foto(Origine).inizioPixel
    .header = Foto(Origine).header
    .g = Foto(Origine).g
  End If
End With
End Sub
```

Per l'applicazione di filtri puntuali si veda il seguente estratto di subroutine che applica la trasformazione in negativo della foto corrente. Si devono modificare tutte le componenti della matrice dei colori dei pixel.

```
Private Sub FiltraFoto(ByVal TipoFiltro As String)
Dim i As Long
Dim j As Long
Dim n As Long

With Foto(gCorrente)
  n = 3 * .w
  For i = 1 To .h Step 1
    j = 1
    Do While j <= n
        Select Case TipoFiltro
          Case "Negativo"
            ' evidentemente si trattano allo stesso modo
            ' le tre componenti componente blue
            .g(j, i) = CByte(255 - .g(j, i))
            j = j + 1
                  ' componente green
            .g(j, i) = CByte(255 - .g(j, i))
            j = j + 1
                  ' componente red
            .g(j, i) = CByte(255 - .g(j, i))
            j = j + 1
          Case ...  ' per gestire altri filtri
            ...
        End Select

    Loop
  Next i
  ' salvo la foto in un file temporaneo
  Open "temp.bmp" For Binary As #1
  Put #1, , .header
  Put #1, , .g
  Close #1
  ' visualizzo il risultato
  Set .FormFoto.picFoto.Picture =
                                LoadPicture("temp.bmp")
End With

End Sub
```

CONCLUSIONI

Nella seconda parte di questo articolo verranno affrontate le trasformazioni locali dell'immagine, le quali consentiranno di agire sul contrasto della stessa, di attenuarne il rumore elettronico ed anche di estrarre i contorni degli oggetti in essa rappresentati.

Roberto Bandiera

Visual Basic

Fotolab

Filtri Fotografici in Visual Basic

Riferimenti

Per approfondimenti e una ampia rassegna di tecniche di elaborazione di immagini si visiti il sito del Dipartimento di Intelligenza Artificiale dell'Università di Edinburgo
http://www.dai.ed.ac.uk/HIPR2/hipr_top.htm

Presso il Dipartimento di Computer Science della stessa università sono reperibili informazioni sui formati grafici
http://www.dcs.ed.ac.uk/home/mxr/gfx/2d-hi.html

Per una introduzione alla fotografia digitale si consulti il manuale on line "Le Tecniche Fotografiche In Archeologia" di Fausto Gabrielli del Dipartimento di Scienze Archeologiche – Università di Pisa
http://www.arch.unipi.it/Foto_Libro/Foto_Libro.html

Animazione IN JAVA 3D (parte terza)

J2SE 1.4

Dopo aver toccato i temi fondamentali per quanto riguarda la creazione dei nostri universi tridimensionali, possiamo ora addentrarci in quegli argomenti, più marginali, che sono utili per approfondire le nostre conoscenza delle librerie Java per lo sviluppo 3D. E quale tema è più interessante trattare se non l'animazione delle scene che abbiamo imparato a generare? Vediamo quindi come procedere, una volta create le scene che intendiamo animare, per fare in modo che esse risultino dotate di una qualche dinamicità.

File Sul CD
soft\codice\allegati3.zip

Innanzitutto occorre fare una importante premessa, che riguarda il modo in cui possono essere generate le animazioni. Esse infatti possono avere due diverse origini: una data dallo scorrere del tempo, (nel qual caso si parla di *animazioni*), ed una invece dall'interazione dell'utente con la scena, (denominate appunto *interazioni*). Anche se la logica con la quale questi due differenti metodi funzionano è simile, le classi sulle quali si appoggiano sono differenti, rendendo di fatto diverso il modo in cui è necessario procedere per ognuno dei due comportamenti. Ma vediamo appunto come le API 3D ci vengono in aiuto nella programmazione di scene dinamiche, illustrando i passi fondamentali da compiere per arrivare alla creazione di un esempio pienamente funzionante.

ANIMAZIONE E INTERAZIONE

Abbiamo detto che le due differenti forme di dinamicità possibili, poggiano comunque sulle stesse basi, vediamone quindi i punti in comune. Ogni azione dinamica in Java3D si realizza utilizzando i *Comportamenti (Behaviors)*. Un Comportamento non è altro che una classe che si occupa di apportare delle modifiche ad un oggetto o alla scena stessa in risposta a particolari eventi che si sono scelti di intercettare. Occorre quindi, per poter realizzare una scena dinamica, creare dei Comportamenti che possano essere sfruttati per modificare il nostro universo. Come fare per creare un comportamento? I passi necessari sono pochi e semplici, infatti si deve:

1. Creare una classe che estenda la classe *Behavior*.
2. Fornire la classe di un costruttore che riceva l'oggetto da modificare.
3. Specificare una condizione di attivazione del Comportamento.
4. Specificare quali modifiche il comportamento deve apportare all'oggetto.
5. Specificare la prossima condizione di attivazione.

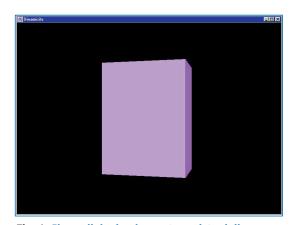



Fig. 1: **Il parallelepipedo, protagonista della nostra scena.**

Anche se i passaggi non sono molti, è comunque utile sapere che vi sono numerose classi predefinite che si occupano di rispondere agli eventi più comuni, e che ci sollevano dal gravoso compito di dover scrivere un Comportamento per ciascuna differente animazione o interazione. Vedremo più in là queste classi e come è possibile utilizzarle per i propri scopi, ma, per ora, dedichiamoci allo studio del funzionamento dei Comportamenti analizzando passo passo tutti i punti necessari per la realizzarne uno personalizzato. A tal scopo analizziamo il listato seguente:

```
public class ComportamentoSemplice extends Behavior{
  private TransformGroup oggetto;
  private Transform3D rotazione = new Transform3D();
  private double angolo = 0.0;
  public ComportamentoSemplice(TransformGroup
                                        oggetto){
```

Navigazione

Per navigare nell'universo che abbiamo realizzato, occorre utilizzare i seguenti tasti:

Freccia su = zoom in;
Freccia giù = zoom out;
Freccia destra = spostamento a destra;
Freccia sinistra = spostamento a sinistra;
Pag. su = spostamento in alto;
Pag. giu = spostamento in basso;

Se non riuscite a muovervi nella scena, potrebbe essere necessario fare click sulla finestra dell'applicazione in modo da selezionarla come attiva.

J2SE 1.4

Animazione IN JAVA 3D

```
        this.oggetto = oggetto;}
    public void initialize(){
        this.wakeupOn( new WakeupOnAWTEvent (
                                KeyEvent.KEY_PRESSED)); }
    public void processStimulus( Enumeration parametri ){
        angolo += 0.1;
        rotazione.rotY(angolo);
        oggetto.setTransform(rotazione);
        this.wakeupOn( new WakeupOnAWTEvent (
                                KeyEvent.KEY_PRESSED));}
}
```

Come appare chiaro dal listato, implementare Comportamenti personalizzati non è eccessivamente complicato anche se, nel caso di procedure più complesse, il codice può risultare sicuramente più impegnativo. Occorre premettere che questo comportamento fa in modo che un solido ad esso associato, una volta visualizzata la scena, rimanga immobile finché l'utente preme un tasto. Quando un tasto viene premuto, esso inizia a roteare su se stesso intorno all'asse delle *Y* senza più fermarsi. Analizzando la dichiarazione della classe, la prima cosa da notare è che essa deriva dalla classe *Behavior*, come specificato al punto 1 del precedente elenco. Seguono poi una serie di proprietà private della classe tra cui distinguiamo *oggetto* che rappresenta l'oggetto che il comportamento andrà a modificare. Il costruttore della classe, che rappresenta il 2° punto della nostra scaletta, non fa altro che impostare il valore di questa variabile. Arriviamo finalmente al primo dei metodi ereditati dalla classe *Behavior*, soprascritto per dotare la nostra classe di una qualche nuova funzionalità: *initialize()*. Questo metodo viene richiamato dal sistema in risposta ad un evento che corrisponde all'iniziazione del Comportamento, che si verifica quando una scena viene visualizzata. Il metodo posto all'interno della funzione, di nome *wakeupOn(…)*, si utilizza per dichiarare al Comportamento di rimanere in attesa della condizione specificata, esaudendo così il punto 3 del nostro elenco. La condizione utilizzata nel nostro esempio altri non è che l'evento di pressione di un tasto. La funzione successiva, *processStimulus(…)*, è anch'essa ereditata da *Behavior*, ed è il metodo che viene richiamato quando si verifica la condizione di attivazione del Comportamento, cioè nel nostro caso la pressione di un tasto. In questo metodo è possibile implementare il punto 5 della nostra scaletta, visto che è proprio questa la funzione che modifica effettivamente le varie geometrie del solido associato al Comportamento. Le istruzioni che formano il corpo della funzione, sono infatti le istruzioni di modifica dell'angolazione sull'asse *Y* del solido. Per finire, come nel punto 6 dell'elenco, è necessario reimpostare una condizione di attivazione per fare in modo che il solido richiami questa stessa funzione al verificarsi di un nuovo evento. Il metodo *wakeupOn(…)*, infatti, termina la sua funzione dopo la prima volta e se non si imposta nuovamente il Comportamento in attesa di un evento, esso terminerà di essere utilizzabile. Bene, il nostro comportamento è pronto per essere utilizzato ma, per farlo, è necessario definire ancora un paio di cose. Innanzitutto affinché il Comportamento funzioni è necessario applicargli una sfera di influenza, che corrisponde alla distanza entro la quale il Comportamento viene attivato. In parole semplici, se siamo molto distanti da un oggetto che deve subire delle modifiche, è probabile che dette modifiche non risultino visibili all'utente ma, ciononostante, esse saranno state eseguite, portando via tempo di calcolo prezioso in un sistema di rendering in tempo reale. Per evitare questo spreco, è stato pensato un sistema di gestione delle animazioni che prende in considerazione soltanto quelle la cui sfera di influenza racchiude la vista attualmente utilizzata dall'osservatore. Un altro importante punto da rispettare affinché le animazioni che abbiamo pensato funzionino a dovere, è quello di impostare il Trasformatore incaricato di modificare uno o più oggetti. A questo scopo occorre impostare per quest'ultimo la possibilità di essere modificato in uno o più modi. Ma vi sarà tutto più chiaro quando leggeremo il listato della nostra applicazione.

Fig. 2: Visti da vicino.

Comportamenti

Oltre a quelli visti nel listato di esempio, sono molte le classi che si occupano di fornire dei Comportamenti predefiniti utili per i casi più semplici. Tra queste sicuramente le più utilizzate sono *MouseBehavior*, che serve per gestire oggetti tramite mouse, e tutte le sottoclassi di *Interpolator*, utili per modificare forma, colore, dimensione e trasparenza di un solido.

IL NOSTRO ESEMPIO

Questa volta, visto che ormai dovreste essere in grado di capire molti dei passaggi riportati, eviterò di commentare il superfluo, facendovi soffermare esclusivamente sui punti che formano l'argomento odierno. Ma prima di tutto analizziamo il codice che forma la nostra applicazione:

```
import java.awt.*;
import javax.swing.*;
import javax.vecmath.*;
import javax.media.j3d.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
public class Dinamicita extends JFrame{
  public Dinamicita(){
     super("Dinamicita");
     setSize(800,600);
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    JPanel nuovoPannello = new JPanel();
    nuovoPannello.setLayout(new BorderLayout());
    Canvas3D tela = new Canvas3D(null);
    nuovoPannello.add("Center",tela);
    SimpleUniverse universo = new SimpleUniverse(tela);
    universo.getViewingPlatform()
                          .setNominalViewingTransform();
   BranchGroup gruppo = new BranchGroup();
   // Inseriamo qui il codice per costruire la nostra scena 3D
   /*** ANIMAZIONE ***/
    // Iniziamo creando un oggetto che si occupi dello
                                       scorrere del tempo
   Alpha andatura = new Alpha(-1,5000);
    // Poi creiamo un'area entro la quale le animazioni
                                            hanno luogo
   BoundingSphere area = new BoundingSphere();
    // Ora costruiamo tre strutture che identifichino i
                                         colori primari...
   Color3f rosso = new Color3f(1.0f,0.0f,0.0f);
   Color3f verde = new Color3f(0.0f,1.0f,0.0f);
   Color3f blu   = new Color3f(0.0f,0.0f,1.0f);
  // con i quali costruiamo poi un materiale...
  Material materiale = new Material(blu,rosso,verde,
                          rosso,0.0f);  //modificabile...
   materiale.setCapability(
                   Material.ALLOW_COMPONENT_READ|
                  Material.ALLOW_COMPONENT_WRITE);
  // che utilizziamo per costruire un'estetica...
  Appearance estetica = new Appearance();
  estetica.setMaterial(materiale);
  // necessaria a creare infine il nostro solido.
  com.sun.j3d.utils.geometry.Box solido = new
  com.sun.j3d.utils.geometry.Box(0.2f,0.4f,0.3f,estetica);
  // Creiamo ora il trasformatore da utilizzare per le
                                     nostre animazioni...
  ...
  /*** ILLUMINAZIONE ***/
  // Illuminiamo adeguatamente la scena per renderla
                                                visibile
  ...
  // Ora impostiamo i comandi per rendere il nostro
                                        mondo navigabile
 /*** INTERAZIONE ***/
 ...
public static void main(String[] args){
    new Dinamicita(); }
}
```

ANIMAZIONE

Il listato di esempio riportato si occupa di creare un parallelepipedo applicandovi un materiale non predefinito, nonché di animarlo, facendolo roteare su se stesso, e modificarne i valori del materiale associato in modo che sembri cambiare colore. Inoltre, al punto di vista utilizzato per generare la scena, viene applicato un Comportamento che rende possibile la navigazione della stessa.
Sarà quindi possibile, utilizzando dei tasti particolari, muoversi all'interno della scena grafica e gironzolare intorno al nostro parallelepipedo animato. Il codice fino al commento "Inseriamo qui il codice per costruire la nostra scena 3D" è stato più volte analizzato e serve a generare un'applicazione che contenga il nostro universo. Subito dopo appare il primo dei blocchi di codice che ci interessa, quello relativo all'animazione della scena.
La prima istruzione che troviamo crea un oggetto *Alpha*, che serve per essere associato a dei Comportamenti che devono verificarsi con lo scorrere del di tempo. In realtà l'oggetto Alpha è molto potente e configurabile, e permette di stabilire esattamente: *quando attivare* i Comportamenti ad esso associati, *quante volte* farlo nel corso dell'applicazione e *dopo che lasso di tempo*.
Quest'oggetto è uno dei più importanti per generare animazioni dovute al passaggio del tempo. Nella riga successiva troviamo un altro oggetto che dovremmo aver imparato a conoscere: *BoundingSphere*. Esso si occupa di fornire un'area entro la quale tutti gli oggetti ad esso associati (e come potete vedere sono molti) compiano le azioni per la quali sono stati programmati. Come infatti già spiegato, le animazioni e le interazioni, vengono calcolate esclusivamente se la loro sfera di influenza (*BoundingSphere*) risulta contenere anche il punto di vista utilizzato dall'utente per renderizzare la scena. Il costruttore di default utilizzato nel nostro esempio compie egregiamente il suo lavoro, impostando dei limiti più che sufficienti alle nostre esigenze.
Successivamente Il listato crea tre strutture dati che identificano i tre colori primari, rosso verde e blu, necessari a creare un oggetto di tipo *Materiale*. Sin ora non ci siamo mai occupati della realizzazione di solidi che avessero delle caratteristiche estetiche diverse da quelle predefinite ma, in questo caso, creeremo una figura che ha delle proprietà visive uniche. I parametri passati per la creazione del *Materiale*, infatti, rappresentano come esso risponde visivamente quando interessato dalle sorgenti luminose. Ma a cosa ci serve?
Come preannunciato, le animazioni di una scena grafica possono andare a modificare numerosi campi di un solido, tra cui anche il suo materiale. Ma per farlo, quest'ultimo deve essere predisposto ad accettare eventuali modifiche. Ed è proprio a questo che serve la riga di codice successiva, che fa in modo che il nostro materiale risulti modificabile in lettura e scrittura. Una volta generato questo Materiale modificabile esso viene utilizzato, nelle due linee successive, per creare un oggetto di tipo *Appearance* che rappresenta l'insieme delle specifiche estetiche di un solido. Oltre al materiale, infatti, esso può avere associate diverse caratteristiche (textures, trasparenze, etc.etc.) che vengono immagazzinate in un unico oggetto che funziona da contenitore per tutte le possibili caratteristiche estetiche di uno o piu solidi. Nel nostro caso, l'unica particolarità dell'oggetto *Appearance* sarà nel materia-

J2SE 1.4

Animazione
IN JAVA 3D

WAKE_UP_EVENTS

Molte sono anche le condizioni che possono essere implementate nei Comportamenti per fare in modo che questi ultimi vengano attivati in un particolare momento. Si può apportare una modifica alla scena grafica ad esempio quando la vista dell'utente entra nell'area di influenza del Comportamento, oppure in seguito a delle collisioni, o in base ad un'evento generato dall'utente, etc. etc.

J2SE 1.4

Animazione
IN JAVA 3D

le che abbiamo creato.
Fatto ciò, non rimane altro da fare, nella successiva linea di codice, che generare il nostro solido con associato l'oggetto *Appearance* appena realizzato. Ora che abbiamo un oggetto visivo sul quale lavorare, il listato prosegue andando a creare un oggetto TransformGroup che ci servirà per compiere le nostre animazioni. Come introdotto nel primo articolo di questa serie, infatti, un qualsiasi oggetto da aggiungere alla scena grafica va aggiunto ad un Gruppo principale, da collegare poi alla locale fornitaci dalla classe SimpleUniverse che identifica il nostro universo. Ma se si progetta di compiere delle trasformazioni su uno o più oggetti grafici, allora è necessario introdurre tra il gruppo ed il solido un altro oggetto di tipo TransformGroup, utile a realizzare la trasformazione. Per fare in modo che questo oggetto sia in grado di apportare delle modifiche alla scena anche dopo che questa sia stata compilata, si dono impostare delle capacità al trasformatore così come fatto nella linea di codice successiva alla creazione dello stesso. Dopo questa istruzione il trasformatore sarà in grado di modificare il o i solidi ad esso associati.
Proseguendo nella lettura del listato, si notano i due successivi blocchi di codice compiere azioni molto simili. Essi altro non sono che dei Comportamenti predefiniti di Java3D che noi utilizziamo per dichiarare in modo il nostro solido deve venire modificato.
La classe ColorInterpolator si occupa di modificare il materiale associato al nostro oggetto grafico, ed infatti in fase di costruzione dello stesso gli vengono passati come parametri il materiale da modificare e l'oggettoAlpha che indica con che ritmo e per quante volte apportare la modifica.
La classe RotationInterpolator, invece serve a generare un Comportamento di rotazione per qualsiasi oggetto le si associ. Come è possibile osservare dal codice, l'oggetto passato al costruttore di RotationInterpolator, insieme al già visto oggetto Alpha anche qui necessario, è il Trasformatore precedentemente creato ed al quale assoceremo successivamente il nostro solido.
Se inserissimo più solidi nel nostro trasformatore, essi subirebbero tutti la medesima animazione. Entrambi i blocchi di codice continuano poi, dopo l'utilizzo del costruttore per generare due oggetti di tipo Comportamento che compiono animazioni differenti, impostando i limiti di influenza di quel Comportamento per fare in modo che esso risulti attivo soltanto entro una certa distanza dall'osservatore. Il passo finale poi consiste nell'aggiungere i Comportamenti creati al Trasformatore che abbiamo creato, in modo che esso possa utilizzarli dinamicamente.
I due ultimi passaggi prima del blocco di codice dedicato all'illuminazione, poi, altro non fanno che inserire tra il Gruppo che rappresenta la nostra scena grafica, ed solido che dobbiamo modificare, il Trasformatore, con tutti i vari Comportamenti associati, incaricato della modifica.

Billboard

Esiste anche una tecnica, chiamata appunto Billboarding, che gestisce l'utilizzo di figure bidimensionali al posto di veri e propri solidi, per fare in modo di simulare un'universo solo a prima vista tridimensionale. Java 3D supporta anche questa tecnica.

INTERAZIONE

Prima di arrivare al blocco di codice relativo all'interazione, è necessario passare brevemente in rassegna le istruzioni che rendono possibile la corretta illuminazione della scena. Come già visto nel capitolo dedicato alle luci, le istruzioni in questo blocco creano due delle sorgenti di luce possibili in Java3D, una tenue luce ambientale ed una luce direzionale. Alle luci viene infine assegnata un'area di influenza simile a quela dei Comportamenti.
Finalmente arriviamo al gruppo di istruzioni che rendono possibile la navigazione del nostro universo tridimensionale. Come potete osservare, il listato è pressoché identico a quello degli altri due Comportamenti utilizzati in precedenza, ad eccezione del fatto che qui non è necessario fornire un oggetto *Alpha* che si occupi di innescare gli eventi che modifichino gli attributi del solido. L' inutilità di questo oggetto è data dal fatto che non più lo scorrere del tempo, quanto invece l'input dell'utente, serve da stimolo al Comportamento per essere innescato. Ma vediamo nel dettaglio il funzionamento. Innanzitutto, poiché ciò che intendiamo modificare è la posizione del punto di vista dell'utente, occorre recuperare questo oggetto racchiuso all'interno della classe *SimpleUniverse*. Di questo si occupa la prima linea di codice del blocco dedicato all'interazione. Successivamente, altro non serve che procedere come visto prima per gli altri due Comportamenti, e cioè costruendo un oggetto di tipo Comportamento che compia le azioni che ci interessano, impostarne i limiti di influenza ed aggiungendolo alla scena. Le uniche differenze qui sono date dall'assenza dell'oggetto *Alpha* e dall'aver aggiunto il Comportamento non al Trasformatore, poiché esso deve occuparsi solo della modifica del solido, bensì all'oggetto che lo contiene, cioè il Gruppo che andremo poi ad aggiungere all'universo. Fatto ciò non resta che terminare il programma come fatto più volte, associando il gruppo all'universo, compilando e mostrando il tutto.

CONCLUSIONI

Gli argomenti trattati oggi sono veramente solo una breve introduzione su un argomento vastissimo che meriterebbe un libro intero. È infatti possibile, utilizzando animazioni ed interazione, arrivare a risultati incredibili in un prodotto immediato e potente come Java3D. Gli strumenti presenti nelle API permettono infatti profonde manipolazioni, effetti di morphing, sostituzioni di oggetti, textures, etc.etc. Starà a voi, alla vostra fantasia e voglia di sperimentare, arrivare a generare ambienti tridimensionali animati ed interagibili che possano essere utilizzati nelle vostre applicazioni. A me non resta che augurarvi di riuscire sempre ad orientarvi in un mondo affascinante e complesso come quello della programmazione grafica tridimensionale.

Giuliano Uboldi

Biblioteca

ON LINE

DevShed

Un sito interamente dedicato alla comunità degli sviluppatori. Si distingue per i frequentatissimi forum messi a disposizione (ASP, XML, Perl, Visual Basic, Flash, C++, ecc, ecc).

http://www.devshed.com/

DevBuzz

DevBuzz.COM è un sito dedicato a chi si cimenta giorno per giorno nella programmazione dei Pocket PC. Mette a disposizione una vasta serie di tool, tecniche di programmazione e soluzioni a svariate tipologie di problemi.

http://www.devbuzz.com

.NET 247

Per dirla così come si auto-sponsorizza il sito: il primo sito dedicato al framework .NET, interamente indipendente. .NET problem? .NET answer.

http://www.dotnet247.com/

247reference/default.aspx

Flash MX – Tutto & Oltre

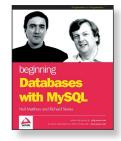
La collana Tutto & oltre di Apogeo, punto di riferimento degli utenti avanzati e professionali, si pregia di un nuovo testo dedicato al linguaggio Flash, vera rivoluzione per il supporto allo sviluppo di pagine Web. Il testo mostra passo passo come sfruttare al meglio il più potente programma di animazione vettoriale che ha rivoluzionato il modo di progettare e creare pagine Web. Ricco di illustrazioni, esempi e dimostrazioni pratiche a cura di esperti del settore, è sicuramente un libro che non può mancare nella biblioteca di chi desidera costruire lavori di qualità. Tra gli argomenti trattati:

- Introduzione all'uso di Flash
- I nuovi strumenti di disegno
- Contenuto caricato dinamicamente nei progetti Web
- Utilizzo di Action Script e dei componenti precostituiti
- Effetti dinamici, audio MP3, giochi interattivi in Flash

Nel CD-ROM allegato: software in versione trial e shareware, aggiunte, plug-in, modelli e file degli esempi trattati nel testo.

Difficoltà: Medio-Alta • Autori: Reinhardt Robert, Dowd Snow • Editore: Apogeo http://www.apogeonline.com • ISBN: 88-503-2070-1 • Anno di pubblicazione: 2002 Lingua: Italiano • Pagine: 1048 • Prezzo: € 55,00 • Contiene 1 CD-Rom

Beginning database with MySQL

MySQL rappresenta il prodotto più popolare tra gli RDBMS Open Source, rinomato soprattutto per velocità e potenza di elaborazione. Il testo si propone come guida di riferimento alla scoperta del database, analizzandone, approfonditamente, tutte le funzioni più importanti. Un tour alla scoperta di MySQL, che guida il lettore, passo passo, dall'installazione alla sua configurazione, all'esecuzione di comandi, semplici e complessi, sia per l'amministrazione del sistema che per il suo utilizzo pratico. Molti esempi mostrano come integrare la potenza di MySQL all'interno dei più comuni linguaggi di programmazione orientati al Web.

Difficoltà: Medio-Alta • Autori: Richard Stones, Neil Matthew • Editore: WROX http://www.gorilla.it • ISBN: 1-861006-92-6 • Anno di pubblicazione: 2002 • Lingua: Inglese Pagine: 608 • Prezzo: $39,99

Building an ASP.NET Intranet

Il libro giusto per tutti coloro che vogliono apprendere i concetti di base per lo sviluppo di applicazioni Intranet, utilizzando le nuove funzionalità messe a disposizione dalla tecnologia ASP.NET. In questo testo gli autori hanno posto l'accento più sull'aspetto pratico che teorico, mostrando, di fatto, un esempio reale di applicazione, nella fattispecie IBuySpy Portal.

- Qual è la differenza tra un'applicazione Intranet e un'applicazione Internet?
- Quali i requisiti per creare un'applicazione Intranet?
- Come gestire la sicurezza di un'applicazione Intranet?

Queste alcune delle domande che trovano risposta nel testo.

Difficoltà: Medio-Alta • Autori: Vari • Editore: WROX http://www.gorilla.it • ISBN: 1-861007-49-3 Anno di pubblicazione: 2002 • Lingua: Inglese • Pagine: 480 • Prezzo: $49,99

Tips&Tricks

I trucchi del mestiere

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca sulla Rete delle Reti, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. **Il codice completo dei tips lo trovate nel CD allegato nella directory \tips\.**

Visual Basic.NET

Come sfruttare

l'ereditarierà visuale in Visual Basic .NET

Visual Basic ha sempre avuto come punto di forza una estrema semplicità nella costruzione delle interfacce utente, il punto in cui pagava dazio ad altri linguaggi come C++ e Java era lo scarso supporto alle più avanzate funzionalità Object Oriented. Questo gap è stato ampiamente superato con Visual Basic.NET. Nell'esempio che andiamo ad illustrare vedremo proprio come sfruttare l'ereditarietà nelle interfacce utente.

Supponiamo di voler distribuire un'applicazione in cui tutte le form presentino il marchietto dell'azienda committente in alto a destra. Grazie alla ereditarietà potremo creare una volta per tutte la struttura base della form ed ereditarla in tutte le altre. Inoltre, se ci troveremo a distribuire la stessa applicazione per un'altra azienda, potremo cambiare il marchio stesso o la sua posizione, ottenendo che al modifica si ripercuota a cascata in tutte le form.

CREIAMO LA CLASSE BASE

Ricordando che anche una form altro non è che una classe, apriamo un nuovo progetto in Visual Studio .NET e creiamo una nuova libreria di classi: avere qui la nostra forma base ci consentirà di aggiungerla successivamente in qualsiasi applicazione sviluppata in seguito. Chiameremo questa libreria *LibreriaForm*. In questa libreria andremo ad aggiungere una nuova form, chiamiamola *ioForm* (Fig. 1).

Fig. 1: Aggiunta di un nuovo elemento.

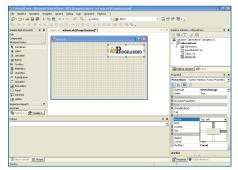



Fig. 2: Interfaccia dell'applicazione.

Per aggiungere il logo utilizziamo una PictureBox, andando a indicare il percorso in cui si trova l'immagine relativa. Posizioniamo la PictureBox in alto a destra e andiamo a individuare fra le proprietà quella relativa all'ancoraggio. Per default, l'ancoraggio è fissato in alto a sinistra, noi lo sposteremo in altro a destra. Questo accorgimento consentirà di tenerlo nell'angolo, alla corretta distanza dai bordi, qualsiasi sia la dimensione della finestra (Fig. 2). Quando in seguito deriveremo una form dalla form base, tutti i

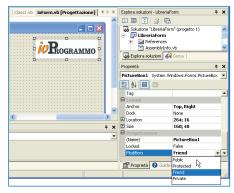



Fig. 3: Modifier

controlli che abbiamo sistemato nella form di libreria saranno presenti anche nella form derivata. Le proprietà di questi controlli non potranno essere modificate se i corrispondenti controlli nella form base hanno la proprietà *Modifier* impostata a *Private* o *Friend*. Per lasciare alle classi derivate la possibilità di modificare le caratteristiche dei controlli è dunque necessario impostare la proprietà *Modifier* dei controlli a *Public* oppure *Protected*. Andiamo ad operare in questo modo sul controllo relativo al logo.

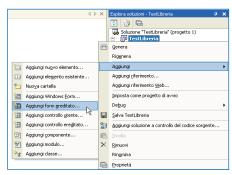



Fig. 4: AggiungiFormDerivata

CREIAMO LA FORM DERIVATA

Prima di poter sfruttare la libreria a cui stiamo lavorando, è necessario compilare il progetto. E' bene tenere presente che, al fine di rendere effettive le modifiche, è necessario ricompilare il progetto ogni volta che andiamo a modificare la form base. Per provare la libreria appena creata, imposteremo un nuovo progetto ma, ovviamente, la nostra *Libre-*

riaForm può essere utilizzata da qualsiasi progetto .NET, anche già esistente. Il nuovo progetto lo chiameremo *TestLibreria* e lo creiamo come applicazione VB.NET. Andiamo dunque ad aggiungere una form ereditata, cliccando con il tasto destro sul nodo *TestLibreria* di *Esplora Soluzioni*. Se la form da cui vogliamo derivare si trova in un progetto diverso da quello corrente, dovremo specificare il percorso in cui si trova la DLL relativa alla classe base: nel nostro caso *\LibreriaForm\bin*. Così facendo avremo nel nostro progetto una form del tutto identica a quella presente in libreria. Se guardate con attenzione il controllo relativo al logo, potete notare una piccola freccia in alto a sinistra. La freccia sta a indicare che il controllo è stato ereditato.

Fig. 5: Notate la piccola freccia in alto a sinistra.

MODIFICHE

Grazie al fatto che abbiamo dichiarato public la proprietà Modifier della classe base, possiamo aggiornare l'aspetto ed il comportamento della classe derivata. Proviamo, ad esempio, a cambiare la posizione del logo, spostandolo nell'angolo in alto a sinistra e modificando la relativa proprietà di ancoraggio con "Top, Left". A questo punto possiamo anche aggiornare il logo con uno più recente, modificando la proprietà Image…

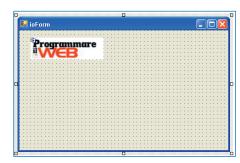



Fig. 6: Risultato finale.

Nella figura trovate il risultato dei nostri sforzi: l'ereditarietà visuale è servita! Sul CD potete trovare i progetti in Visual Studio .NET relativi all'esempio presentato.

C++

Come risolvere equazioni di secondo grado

Una piccola applicazione che si occupa di risolvere equazioni quadratiche del tipo $Ax^2 + Bx + C = 0$ e che implementa due funzioni specifiche: *numRoots(a, b, c)* e *quadSolve(a, b, c)*; la prima determina il numero di radici, mentre la seconda si occupa di trovare le radici.

Come leggere un file riga per riga

Utilizzando le *Standard Template Library*, questo piccolo codice consente di accedere ad un file e leggerlo riga per riga. Per immagazzinare i dati così ottenuti viene utilizzato un vettore: ogni riga è memorizzata in una cella di questo vettore.

Come calcolare il numero di nepero

Utilizzando le serie di *Taylor*, questo piccolo pezzo di codice si occupa di calcolare il numero di Nepero. Non particolarmente veloce, ha nella semplicità la sua forza.

Come ridurre lo sfarfallio durante il ridimensionamento delle finestre

Come tutti sanno, il messaggio *WM_PAINT* è inviato ad una finestra ogni qual volta si rende necessario ridisegnarla. Dunque, generalmente, scriviamo tutte le istruzioni relative al layout della finestra nell'handler di *WM_PAINT*, incluse le istruzione relative allo sfondo. Quello che accade in realtà è che prima di *WM_PAINT* è inviato il messaggio *WM_ERASEBKGND*, la cui implementazione di default cancella l'area corrispondente alla finestra con un rettangolo bianco. Per migliorare l'efficienza può essere utile individuare gli oggetti che non cambiano sullo schermo, il codice che si occupa di disegnare questi oggetti può essere dunque spostato dall'handler di *WM_PAINT* a quello di *WM_ERASEBKGND*, avendo l'accortezza di ritornare un valore diverso da zero per indicare che non è più necessaria la cancellazione riferita a quell'oggetto.
Con questa soluzione si riduce la complessità delle operazioni di disegno di una finestra e si elimina l'effetto di sfarfallio durante lo spostamento ed il ridimensionamento delle finestre.

Java

Come assegnare la dimensione di un array a run-time

Grazie al fatto che in Java un array non è altro che un oggetto, la sua dimensione è trattata come una proprietà dell'oggetto stesso e può dunque essere assegnata a run time. Questa caratteristica diviene di fondamentale importanza in tutti i casi in cui abbiamo bisogno di un array per immagazzinare dei dati ma non possiamo conoscere la dimensione esatta prima di eseguire l'applicazione.

Come implementare una funzione di search&replace

Una classe semplice ed utile che consente di effettuare la classica funzione di "trova e sostituisci" all'interno di una stringa. Il metodo riportato sul CD accetta tre parametri: la stringa su cui effettuare la ricerca, la stringa di caratteri da cercare e la stringa di caratteri che andrà a sostituire quella originale.

Come semplificare l'utilizzo degli switch

I programmatori C++ sono abituati ad analizzare gli argomenti passati dalla linea di comando attraverso i parametri "*int argc*" e "*char *argv[]*" della funzione *main()*. Un utilizzo del tutto simile dei parametri è possibile attuarlo in Java per impostare le opzioni di avvio di un'applicazione. La classe presente sul CD rappresenta un utile framework su cui basare una propria implementazione degli switch.

Come caricare una classe da un file Jar

Una semplice, benchè completa, implementazione di un class loader per recuperare delle classi java da un file JAR durante l'esecuzione di un'applicazione. Il codice allegato, ampiamente commentato, può essere facilmente personalizzato al fine di essere integrato nei nostri progetti.

Come sfumare lo sfondo

Con il codice qui presentato avremo a disposizione un metodo semplice ed efficace per ottenere il classico effetto di sfumatura dello sfondo tipico, ad esempio, di molte applicazioni di installazione. La sfumatura andrà a schiarirsi dal basso verso l'alto, mentre la scelta del colore base si effettua attraverso la combinazione dei tra parametri booleani accettati dalla funzione (rosso, verde e blu). Ad esempio, per avere la sfumatura blu come sfondo è sufficiente scrivere il seguente codice nella procedura *Form_Load*:

```
FadeForm Me, False, False, True
```

Come mostrare i font durante la selezione

La prima cosa da fare è popolare la ComboBox con i nomi dei font disponibili nel sistema, per fare ciò è sufficiente aggiungere il seguente codice a *Form_Load()*

```
    For I = 0 To Screen.FontCount - 1
        ' Put each font into list box.
        cboFont.AddItem Screen.Fonts(I)
    Next I
```

Non resta che gestire l'evento *Click* aggiungendo il codice che imposta il font del ComboBox:

```
Private Sub cboFont_Click()
    'Set the FontName of the combo box
    'to the font that was selected.
    cboFont.FontName = cboFont.Text
End Sub
```

Come salvare la dimensione e la posizione di una Form

Molti di voi avranno notato che, all'avvio, alcune applicazioni "ricordano" la posizione e la dimensione in cui si trovavano al momento dell'ultimo utilizzo. In questo tip spieghiamo come realizzare lo stesso effetto in un'applicazione Visual Basic. In un modulo dell'applicazione specifichiamo una costante di questo tipo:

```
Public Const ApplicationName = "Nome Applicazione"
```

Questo nome serve a distinguere l'applicazione nel registro di Windows. Nell'evento *Form_Load* richiamiamo la funzione di recupero delle impostazione, passando come argomento il riferimento alla form corrente:

```
Call LoadFormDisplaySettings(Me)
```

Mentre nell'evento *Form_Unload* richiamiamo la funzione di salvataggio delle impostazioni:

```
Call SaveFormDisplaySettings(Me)
```

Da notare che come effetto indesiderato di questo metodo si ha che nel registro di Windows restano le impostazioni che abbiamo definito anche dopo che l'applicazione viene disinstallata. Se si vuole intervenire su questo aspetto, le informazioni si trovano in:

```
HKEY_CURRENT_USER\Software\VB and VBA Program
                              Settings\My Application Name\.
```

Nel CD trovate il codice delle due funzioni di salvataggio e ripristino delle impostazioni, codice che va inserito in un modulo dell'applicazione.

Come controllare l'input

Capita spesso di utilizzare dei textbox per effettuare l'input di dati di tipo numerico. Per accertarsi che l'utente non digiti caratteri alfabetici è possibile effettuare un controllo a posteriori rispetto all'immissione, anche se sarebbe preferibile impedire direttamente in fase di digitazione l'immissione di caratteri diversi da quelli numerici: proprio quello che consente il codice qui presentato.

▶ Delphi

Come nascondere l'applicazione nella task bar

Per alcune applicazioni può essere utile inibire la visualizzazione del pulsante relativo sulla task bar. Per ottenere questo è sufficiente impostare la proprietà *ShowMainForm* di *Application* a *False*. In Windows 2000 e in Windows XP l'applicazione continuerà a essere visibile tra i processi presenti nel Task Manager: proprio dal task manager sarà possibile fermare l'applicazione, fare in modo che l'applicazione giri in modalità nascosta, la rende assimilabile ad un servizio.

Come utilizzare tipi enumerated come variabili di controllo in un loop

Una caratteristica interessante de cicli *for…do* è quella di poter usare tipi enumerated al posto dei più comuni interi. Utilizzando questa proprietà possiamo incrementare notevolmente il livello di leggibilità del nostro codice, così com'è possibile notare nella porzione di codice riportata sul CD.

Come visualizzare informazioni relative alla occupazione di memoria

Il box "*About*" standard ingloba alcune interessanti informazioni statistiche sulla memoria. In particolare, attraverso il metodo *PhysicalAvMemLbl* è possibile conoscere la quantità di memoria complessivamente disponibile, mentre con *MemoryInUseLbl* si può ottenere la percentuale di memoria fisica occupata.

Come impostare le proprietà comuni a più controlli

Capita spesso di dover settare la medesima proprietà su più controlli: un caso tipico è la proprietà *visible* o anche enabled nelle interfacce. La procedura allegata al CD si occupa di impostate la proprietà visible di un gruppo di oggetti *TControl*. La stessa procedura può essere facilmente modificata ed estesa per accedere ad altre proprierà degli oggetti:

```
SetVisibleProp([NewBtn,OpenBtn,SaveBtn],UserState <> usNotAuthorized);
```

Il Torneo
DI CROBOTS 2K2

Torneo
CRobots 2k2

E dodici. Tanti sono gli anni che ci separano dal primo torneo di crobots, tenutosi nell'ormai lontano 1991, quando la potenza dei computer si misurava in Mhz e l'interfaccia grafica sui P.C. era un lusso che pochi potevano permettersi. Eppure, a dispetto del tempo trascorso, l'attività di assemblare mostriciattoli in Ansi C non smette di attirare nuovi adepti...

...Nonostante il numero dei robot inviati annualmente si mantenga sempre più o meno costante, la scomposizione per autore del parco combattenti mostra caratteristiche sorprendenti: accanto a quei sei o sette veterani, che non saltano una competizione da tempi immemorabili, troviamo, infatti, di volta in volta nomi nuovi che prendono il posto di quanti, per motivi di tempo o di calo di interesse non partecipano più alla manifestazione. Numerosi e graditissimi sono anche i ritorni: autori che, dopo una pausa di riflessione, si ripresentano ai nastri di partenza per sfidare nuovamente la sorte, sia per animati propositi di vittoria, sia per dire, semplicemente, 'eccomi, sono tornato'. Spulciando l'elenco degli autori dei 54 programmi che quest'anno si sono confrontati (per la verità sarebbero di più, ma alcuni sono stati inviati a manifestazione oramai conclusa, un vero peccato), cercando di aggiudicarsi il premio messo in palio da ioProgrammo, ritroviamo con piacere Marco e Luca Pranzo, Tommaso de Prà e Fabio Luciano (confesso di non sapere quale sia il nome), un concorrente che non si vedeva dal lontano 1994. Tra le defezioni segnalo, per l'importanza dell'autore, quella di Dario Serino. A causa degli esami universitari non ha potuto essere dei nostri, ma ha promesso un ritorno in grande stile per il prossimo anno. Con lui in campo, forse, le cose sarebbero andate diversamente. Assai gradita, giunge poi la conferma di Franco Cartieri che, con *Copter2* e *Kyash_2* ribadisce in tutte le competizioni gli ottimi piazzamenti della scorsa stagione. Numerose le matricole, tra le quali si segnala senza dubbio la notevole performance di Antonio Barone che, con il robot *TheSlayer*, caccia molti veterani, tra i quali il sottoscritto, dalle posizioni nobili della classifica: a una prima analisi il suo programma appare lineare ed efficiente.
Senza dubbio fornirà nuovi spunti per la prossima edizione.

I MICROROBOT

Anche quest'anno, come già era accaduto per il precedente evento, la maggior parte dei partecipanti ha, deciso di iscrivere un combattente solo a questa categoria: ben 23 dei 54 sfidanti, infatti, erano al di sotto delle 500 istruzioni. Il girone ha salutato la vittoria di *regis.r* (Daniele Nuzzo) nella modalità *f2f* e di *tigre.r* (Alessandro Carlin) per la specialità 4vs4. Il calcolo combinato delle efficienze che, lo ricordo, attribuisce un rapporto di 5 a 1 ai pesi delle due competizioni, ha decretato l'affermazione finale del piccolo appartenente alla scuderia di Daniele.

IL TORNEO 'CLASSICO'

A questo punto, i microbi, uniti ai 16 mini-robot pervenuti, si sono affrontati per decidere il vincitore nella categoria del crobots 'storico'. Per dovere di cronaca va detto che questi classicbots poco hanno a che spartire con i combattenti scritti nell'epoca delle 1000 istruzioni effettive. Molti di loro, anzi, sono tranquillamente in grado di battere persino i macro robot del torneo 2001. Qui la superiorità del microcampione è stata addirittura imbarazzante: *bruenor.r* ha dominato il primo girone, mentre *regis.r* si è comportato in maniera superlativa anche in presenza di avversari più corpulenti, andando a vincere in solitudine il proprio round. La finale è stata a senso unico, con Bruenor in grado di infliggere un distacco di ben 5 punti percentuali a *Enigma.r*, di Alessandro Carlin (ancora lui!).

Torneo CRobots 2k2

IL TORNEO 2K2

Finalmente si fa sul serio: i primi tre giorni di combattimenti sono stati poco più che un prologo alla manifestazione principale. La domanda aleggia nell'aria: riuscirà qualcuno ad opporsi allo strapotere Nuzziano? La fase eliminatoria incorona vincitori *Drizzt.r* (Daniele Nuzzo), *Yerba.r* e *MoveOn.r* (di Michelangelo Messina). Quest'ultimo si è così confermato il dominatore delle qualifiche: oramai da anni non perde un colpo! Fermo restando che i primi otto qualificati di ogni round hanno passato direttamente il turno, i piazzati dal nono al sedicesimo posto hanno poi dato vita al ripescaggio, che ha fornito i nomi degli 8 robot mancanti alla lista dei 32 finalisti. Finalmente, può prendere il via lo scontro decisivo. Scontata l'affermazione nell'*f2f* di *Wulfgar*, con uno stratosferico 84%. I migliori tra gli altri, *Zorn.r*, *Drizzt.r* e *Obelix.r* si fermano intorno al 74%. Inizia l'ultima battaglia. *Drizzt* si porta subito al comando e, durante il primo 10% del torneo è abbondantemente avanti a tutti, arrivando ad accumulare un vantaggio intorno al 2% su *Zorn*. È praticamente campione. Ma a questo punto succede l'incredibile: *Zorn* inizia una lenta ma costante risalita. Al 25% è dietro solo di un quarto di punto. Al 30% è davanti, seppur di poco. Dal 40% in avanti il vantaggio si stabilizza intorno al 2%. E siamo, infine al 100%. Il calcolo combinato delle efficienze decreta che il campione sia, ancora una volta, e per la seconda volta consecutiva, Alessandro Carlin (che si aggiudica in questo modo la microcamera messa in palio dalla Logitech).

Secondo complessivo è *Drizzt*, che però, nel solo 4vs4, subisce l'onta di essere superato anche da *Rudolf7*, ennesimo esponente della numerosa famiglia di Alessandro. Anche quest'anno impressionanti sono state le performance mostrate dai microrobot nella competizione generale. Daniele trova una parziale consolazione piazzando regis.r all'ottavo posto complessivo, migliore tra gli under 500, e *bruenor.r* al quinto, primo tra gli under 1000. Ma lasciamo ora la parola al campione 2002, che, per inciso, strappa a Daniele anche un altro primato: da quest'anno, infatti, i bicampioni sono ben due.

Pos.	Nome del Crobot	Eff. 4vs4	Eff. f2f	Eff. Totale
1	Zorn	43,79	73,16	48,69
2	Drizzt	41,50	74,48	47,00
3	Rudolf_7	41,51	64,65	45,37
4	Wulfgar	38,15	80,57	45,22
5	Bruenor	36,93	67,23	41,98
6	Yerba	36,35	64,58	41,06
7	MoveOn	35,27	61,50	39,64
8	Regis	31,52	56,80	35,73
9	Enigma	29,46	60,34	34,61
10	TheSlayer	29,35	53,71	33,41
11	Obelix	24,35	70,08	31,97
12	Todos	25,77	51,10	29,99
13	Asterix	24,06	59,45	29,96
14	MedioMan	24,19	50,23	28,53
15	Doom2099	22,09	58,58	28,17
16	Tomahawk	23,41	48,83	27,65
17	Pippo2a	25,36	35,20	27,00
18	Jedi5	21,03	53,26	26,40
19	Padawan	15,84	53,35	22,09
20	Tigre	19,28	34,91	21,89
21	Mind	16,05	49,36	21,60
22	Remus	16,09	48,35	21,47
23	Colosseum	19,37	30,87	21,29
24	Idefix	19,10	28,91	20,74
25	Ska	15,06	39,10	19,07
26	Supernov	19,30	17,38	18,98
27	Stanlio	16,20	24,77	17,63
28	Mazinga	14,12	27,33	16,32
29	Kyash_2	7,56	58,16	15,99
30	Harpo	14,14	22,63	15,56
31	Romulus	11,13	32,71	14,73
32	Dynamite	7,00	23,69	9,78

Tab. 1: La classifica del torneo 2K2.

"Innanzitutto devo dire che sono molto soddisfatto per il successo che ancora una volta ha avuto il torneo e per la sorprendente partecipazione di un congruo numero di matricole che si è unito ai tanti veterani e ad alcuni graditi ritorni. Un torneo (seguito in diretta su internet grazie all'efficenza di Simone) che anche quest'anno si è rivelato combattutissimo e incerto e nel quale non sono mancate le sorprese.
Ad esempio sono rimasto stupito per la notevole performance della matricola Theslayer che, all'esordio, si piazza nella top ten con un codice molto originale e semplice, e dalla straordinaria efficienza di molti robots di piccola taglia capaci di dimostrarsi competitivi con avversari di taglia superiore (primo su tutti Regis), a dimostrazione del fatto che i margini di miglioramento garantiti dalle 2000 istruzioni sono ancora in gran parte da sfruttare. Venendo a Zorn direi che il suo movimento base si può descrivere come una fortunata fusione di Fizban e Rudolf_6 che ottimamente si erano comportati l'anno scorso. Infatti il movimento a "quadrato" del primo è stato trasformato in un rettangolo che, di volta in volta, porta Zorn ad avvicinarsi, e quindi ad attaccare, uno dei robots più vicini. Per quanto riguarda gli scontri 1 contro 1, si tratta di un upgrade della stessa routine di Rudolf_6 con un movimento oscillatorio al centro dell'arena, che in Zorn è però unito ad un movimento rettilineo Nord-Sud.
Questa scelta è stata dettata dalla complementarietà delle due soluzioni: a seconda della situazione Zorn utilizza una routine o l'altra cercando di adattarsi all'avversario che ha di fronte. Un importante aiuto mi è stato dato dal debugger, novità del 2002, che permette di ottimizzare le temporizzazioni del robot. Sono così riuscito a evitare i tempi morti dovuti alla ricarica del cannone e a mantenere sempre una velocità elevata nei cambi di direzione. Non mi rimane che dare appuntamento a tutti al torneo 2003 ma prima ancora in MailingList per discutere e seguire il torneo 91-2k2 tuttora in svolgimento."

Simone Ascheri

Spunti tecnici

Archiviata la pratica del 2002 è iniziato il confronto semi-ufficiale teso a trovare i 32 robot più forti mai scritti, contro i quali allenare i propri sfidanti alla prossima competizione. Mentre scrivo il torneo è ancora in corso di svolgimento: potrete trovare i risultati sul sito ww.itportal.it/crobots

e su go.to/crobots

Il 2002 termina con un verdetto meno contraddittorio di quello fornito dalla passata edizione: si assiste, infatti, alla convergenza delle tattiche vincenti tra microbi, classici e macrorobot.

Ottimizzare
PROGRAMMI DELPHI

Ambienti di sviluppo sempre più complessi e computer di ultima generazione dalle prestazioni strabilianti sembrano aver reso inutile l'ottimizzazione del codice. Vediamo come, quando e perché non abbassare la guardia.

Sono in molti a sostenere, non a sproposito, che l'ottimizzazione sia un'arte. In effetti scrivere del "buon codice" significa anche sapere amalgamare capacità tecniche e creatività, senza dimenticare una generosa dose di passione.
Il know-how necessario per ottimizzare, in modo adeguato, un programma risulta spesso fuori dalla portata del programmatore alle prime armi. E' necessario infatti conoscere a fondo: le caratteristiche dell'architettura hardware, del sistema operativo, del compilatore e del linguaggio di programmazione.
In questo articolo, rivolto principalmente a programmatori che si avvicinano per la prima volta all'argomento, analizzeremo metodi, strumenti e trucchi utilizzabili per aumentare l'efficienza del codice prodotto ed evitare gli errori più grossolani.

CONTRO E PRO

Può sembrare strano, ma non è facile giustificare l'ottimizzazione, in quanto aumentare le prestazioni di un programma costa in termini di:

- **Tempo:** cercare soluzioni migliori significa impiegare del tempo prezioso, è perciò molto improbabile che il codice ottimale si concili con il time-to-market;
- **Stabilità:** i trucchi usati per raggiungere certe prestazioni possono minare il corretto funzionamento del programma;
- **Portabilità e manutenibilità:** l'impiego di alcune ottimizzazioni lega il codice ad un numero relativamente ristretto di architetture e/o compilatori. Inoltre, in assenza di commenti si rischia di ottenere "spaghetti code", ossia sorgenti intricati, poco comprensibili e difficilmente mantenibili.

A volte però il conseguimento di prestazioni elevate rientra tra i requisiti del progetto ed in ogni caso l'efficienza è una discriminante non indifferente: uno dei fattori chiave nella scelta di un programma rispetto ad un altro similare. Considerate poi che nei dispositivi mobili, di strategica importanza per il futuro, le risorse sono limitate e la razionalizzazione del software diventa una necessità imprescindibile.
La "qualità del codice" è direttamente proporzionale all'esperienza del programmatore, la conoscenza delle tecniche di ottimizzazione rappresenta dunque un influente biglietto da visita.
Nel prosieguo dell'articolo verranno illustrate le regole più semplici e generali per ridurre la dimensione dell'eseguibile, dello spazio occupato in memoria e dei tempi di calcolo.

DA DOVE PARTIRE

In generale per ottimizzare un programma è necessario operare (almeno) a due livelli di astrazione:

- **Livello algoritmo/strutture dati:** lo studio della complessità computazionale fornisce una misura oggettiva della "bontà" di un algoritmo e aiuta il programmatore nella scelta del metodo più idoneo alla risoluzione di un problema. E' bene non dimenticare che ogni struttura dati ha delle peculiarità che potrebbero viziare in maniera determinante le prestazioni;
- **Livello implementazione:** solo dopo aver esaminato in modo esaustivo il livello precedente si può pensare di "mettere mano al codice" e cercare i costrutti, le funzioni di libreria ed i trucchi del mestiere che garantiscono la maggiore efficienza.

Entrambi i livelli prevedono una casistica sterminata e come spesso accade i punti di vista sono i più disparati, in alcuni casi nettamente contrapposti, mi limiterò dunque a fornire tecniche applicabili in generale.
Per motivi di spazio trascureremo argomenti importanti come la programmazione parallela (multithreading, presenza di GPU, …) e l'uso delle proprietà dei dispositivi hardware (branch prediction, pipelining, vincoli di accesso alla memoria cache, …). Spero che una volta letto l'articolo termini quali profiler e memory leak risultino meno astrusi.

Sistema

File sul CD

\soft\codice
\Ottimizzazione Delphi

Eseguibili

Per ottenere eseguibili dalle dimensioni ridotte bisogna rinunciare alla VCL e scrivere programmi procedurali basati sulla API di Windows. In alternativa si può utilizzare la libreria open-source KOL/MCK prelevabile da http://xcl.cjb.net
Sul CD è presente *InAndOut*, un programma di soli 6K in grado di posizionarsi nella *traybar* ed aprire/chiudere il lettore CD-ROM.

Sistema

Ottimizzare
Programmi Delphi

EFFICIENZA TEMPORALE

Anche se i programmatori Delphi fanno ampio ricorso a componenti scritti da terze parti, prima o poi sorge, per chiunque, l'esigenza di scontrarsi con l'implementazione di routine a "basso livello".

Fig. 1 Queste impostazioni "suggeriscono" al compilatore di ottimizzare al meglio il codice.

Assert

Durante la fase di debugging/ottimizzazione di un algoritmo è possibile ricorrere alle asserzioni. La procedura *Assert*, descritta nella documentazione di Delphi, valuta un'espressione booleana e nel caso in cui risulti falsa genera un'eccezione di tipo *EAssertionFailed*. Con *Assert* si possono scovare con maggiore facilità bug subdoli: usatela senza parsimonia in fase di sviluppo!

Come tutti i compilatori moderni Delphi prevede una lunga serie di ottimizzazioni automatiche (Fig. 1), vedremo come aiutarlo nel produrre codice performante (Fig. 2).

Fig. 2: Con OllyDbg i curiosi possono studiare il codice prodotto dai compilatori. Nell'immagine un ciclo FOR.

Tra le ottimizzazioni automatiche troviamo:

- **Eliminazione del codice:** il compilatore scarta tutte le istruzioni inutili o comunque non raggiungibili. Le istruzioni considerate utili vengono marcate con un pallino dopo la compilazione, le altre sono completamente ignorate. Per esempio nessuna delle istruzioni della procedura seguente comparirà nell'eseguibile:

```
Procedure Contengo_Solo_Codice_Inutile;
var i,k:Integer;
const c = 0;
begin
if c = 1 then for i:=1 to 100 do k:=k+1;
                        // Istruzione inutile (c=0)
i:=0; // Istruzione inutile
k:=0; // Istruzione inutile
end;
```

- **Variabili nei registri:** esaminando il "tempo di vita" delle variabili Delphi è in grado di memorizzare, senza bisogno di direttive esplicite, le variabili nei registri della CPU. In questo modo vengono drasticamente ridotti i tempi di accesso ed il numero di istruzioni necessarie. Per sfruttare al massimo i registri conviene dichiarare variabili locali e progettare procedure o funzioni con un numero minimo di parametri.
- **Gestione avanzata dello stack:** scrivere funzioni con al massimo tre parametri evita inoltre la creazione di uno stack frame. Le procedure annidate, note anche come *procedure locali*, hanno un impatto negativo sulle prestazioni per via della gestione dello stack: meglio farne a meno.

```
// TODO: mai assegnare un progetto ad un
        programmatore che scrive cose del genere!
Procedure Scarse_Prestazioni (var V:array of char;
              K:integer; var C: char; var R: Real);
var i:Integer;
Function Non_Dovrei_Essere_Qui (A,B,C:Integer;
                        X,Y,Z: Real) : Real;
begin
 Non_Dovrei_Essere_Qui :=(A+B+C)/(X+Y+Z);
end;

begin
 For i:= 1 TO K DO
  V[i]:=C;
 R:=Random;
end;
```

- **Variabili loop induction**: eliminazione delle *common subexpression, loop unrolling, function inlining*, etc. Per avere una descrizione dettagliata, corredata da esempi, delle tecniche elencate fate riferimento al box *Sul Web*.

Se volete migliorare le vostre applicazioni con il minimo sforzo tenete in considerazione che:

- E' bene ridurre il numero di finestre create automaticamente: oltre a rallentare la fase di avvio consumano molta memoria. Nel pannello *Forms* delle opzioni di progetto (Fig. 3) è opportuno spostare, compatibilmente con le esigenze del programma, le form non necessarie nella lista *Available Forms*. Queste ultime, per non incorrere in violazioni di accesso o sprechi di memoria, verranno create e distrutte dinamicamente solo al momento del bisogno:

```
procedure TMiaFinestra.CreaDinamicamente;
var
  MiaFinestra: TMiaForm;
begin
  MiaFinestra := TMiaForm.Create(Application);
  try
    MiaFinestra.ShowModal; // Mostra la finestra modale
  finally
    MiaFinestra.Free; // Libera le risorse
   end;
end;
```

Fig. 3: **Le risorse vanno allocate solo quando servono, gli utenti vi ringrazieranno!**

- I parametri di tipo strutturato o stringa che non vengono modificati all'interno di una procedura /funzione dovrebbero sempre essere preceduti dalla parola chiave *const*, come dimostra l'esempio incluso sul CD, l'incremento delle prestazioni è ragguardevole. Un discorso analogo vale per gli array dinamici o aperti;

```
//Notate la presenza della parola chiave const!
function Scansione_Stringa(const s: string; const c:
                                        char): integer;
var
  i: integer;
begin
  Result := 0;
  // Esamina la stringa s
  for i := 1 to Length(s) do begin
  // Alla prima occorrenza del carattere c esce dal loop
    if s[i] = c then begin
      Result := i;
      break;
    end;
  end;
end;
```

- Le istruzioni di controllo del flusso *abort*, *break*, *continue* e *exit*, nonostante contribuiscano al peggioramento dello stile di programmazione, possono tornare utili per ottimizzare un algoritmo;

- È importante scegliere oculatamente la migliore combinazione tra prestazioni e precisione quando si opera con tipi reali. Come regola generale si dovrebbe evitare l'utilizzo di tipi reali differenti nella stessa istruzione. L'esperienza dimostra che, se possibile, conviene usare *Round* al posto di *Trunc;*

- Per dare una mano al compilatore le selezioni vanno organizzate in maniera intelligente: date la priorità alla condizione più probabile, non dimenticate l'esistenza del costrutto *case* e raggruppate i rami del *case* per valori vicini;

- Le operazioni di *I/O* compiute su dati residenti in memorie di massa devono essere gestite attraverso buffer al fine di minimizzare gli accessi. Una soluzione "sporca" consiste nel caricamento integrale del file in memoria centrale:

```
Procedure Copia_File (const Sorgente,Destinazione:String);
var
  MioFile2: TMemoryStream;
  Buffer: String;
  MioFile1:TFileStream;
begin
   MioFile1:= TFileStream.Create(Sorgente,
                  fmOpenRead); // Apre il file sorgente
  try
    MioFile1.Seek(0, soFromBeginning);
    MioFile2 := TMemoryStream.Create;
    try
      MioFile2.CopyFrom(MioFile1, 0);
      // Letture e scritture avvengono direttamente
                                         nella RAM
      MioFile2.Read(Buffer, SizeOf(Buffer));
      MioFile2.Write(Buffer, SizeOf(Buffer));
      // Per programmi non "didattici" si può usare il
                                   metodo LoadFromFile
    finally
      MioFile2.SaveToFile(Destinazione); // Salva il file
                                            su disco
      MioFile2.Free;
    end;
  finally
    MioFile1.Free;
  end;
end;
```

- Quando le prestazioni sono davvero importanti si possono realizzare routine in linguaggio Assembly ed integrarle nell'applicazione oppure accontentarsi dell'inline assembler. Non è però sempre vero che routine scritte in Assembly risultino più veloci di quelle implementate in Object Pascal, a meno che non sia assolutamente indispensabile state alla larga dalle complicazioni della programmazione a basso livello.

Bottleneck

Letteralmente colli di bottiglia, rappresentano le porzioni di codice che richiedono la maggiore quantità di risorse di calcolo. È fondamentale concentrare i nostri sforzi sull'eliminazione, anche solo parziale, dei colli di bottiglia.

Sistema

Ottimizzare
Programmi Delphi

VCL

La Visual Component Library è la vera responsabile delle dimensioni, non di rado spropositate, degli eseguibili. Grazie alle chiamate di sistema e a librerie come la KOL è possibile ottenere programmi funzionali a partire da 5K.

Ovviamente non ha senso ottimizzare ogni singola linea di codice, ecco perché sono disponibili degli strumenti che misurano i tempi di esecuzione e forniscono statistiche dettagliate sui colli di bottiglia: i *profiler*. Oltre ai numerosi prodotti commerciali esistono anche soluzioni open-source, per i nostri scopi adotteremo il profiler *GpProfile*. Il funzionamento di base è molto semplice: *GpProfile* inserisce delle istruzioni nel vostro progetto, genera un file contenente il resoconto delle misurazioni e al termine dell'esecuzione lo interpreta. I passi da compiere sono essenzialmente quelli riportati di seguito:

1) Aggiungiamo *GpProfile* al menu *Tools* dell'ambiente Delphi;

2) Dopo aver fatto una copia di sicurezza del progetto dobbiamo "instrumentare" il codice con la combinazione di tasti *Ctrl+I*;

3) A questo punto è sufficiente scegliere le routine da sottoporre al cronometraggio, eseguire il programma ed analizzare i risultati presentati da *GpProfile* (Fig. 4);

Fig. 4: L'output di GpProfile al termine dell'analisi, non ignorate le percentuali!

4) Non dimentichiamo di eliminare le istruzioni inserite dal profiler al punto 2 premendo contemporaneamente *Ctrl* e *F2*.

A questo punto possiamo ricavare delle informazioni preziose sulle porzioni di codice che più di altre contribuiscono al degrado delle prestazioni e quindi richiedono un'implementazione ottimizzata.

EFFICIENZA SPAZIALE

Se il nostro obiettivo è ridurre lo spazio occupato, sia in fase di esecuzione sia in termini di dimensioni dell'eseguibile, possiamo utilizzare una serie di metodi più o meno immediati. Distinguiamo innanzitutto le ottimizzazioni delle dimensioni da quelle relative all'occupazione della memoria.
Per le prime possiamo affidarci a:

- **Package**: i package sono normali librerie condivise DLL caratterizzate però dall'estensione *.bpl*. Compilando un progetto con l'opzione *Build with runtime packages* otteniamo un eseguibile dalle dimensioni minime.
  Lo svantaggio principale è che bisogna distribuire anche le librerie impiegate, di conseguenza i package sono realmente utili solo quando vengono condivisi da diversi progetti. Se i package standard venissero forniti con le varie versioni di Windows, gli eseguibili prodotti da Delphi non avrebbero nulla da invidiare a quelli di altri compilatori;

- **Compressori**: se per voi è importante il numero di byte occupati su disco dal programma allora la soluzione migliore è rappresentata dai compressori di eseguibili. Sul mercato ne troviamo per tutti i gusti, tra i più gettonati segnaliamo l'open-source UPX. L'unico problema dei compressori risiede nella gestione della memoria: i programmi compressi generalmente non sfruttano l'allocazione *on demand* ma vengono decompressi completamente nella RAM. Una strada alternativa è rappresentata dalle utility che rimuovono le sezioni superflue, sul CD allegato alla rivista trovate *StripReloc*;

- **Razionalizzazione delle risorse**: l'eseguibile dovrebbe contenere solo le risorse essenziali, eventuali risorse condivise vanno inserite in DLL appositamente create. È buona norma diminuire il numero di colori delle immagini, quasi sempre ne sono sufficienti 256, e caricarle manualmente senza impiegare le scorciatoie messe a disposizione dall'IDE;

- **Clausole unit**: per agevolare lo smart-linking non dovremmo includere unit inutili, inoltre a meno che non ci siano dipendenze mutue, la loro dichiarazione deve avvenire nella *uses* della sezione *implementation* e non in quella della sezione *interface*;

- **Componenti invisibili**: non cedete alla tentazione di effettuare il drag & drop di componenti non visibili in fase di progettazione, l'esiguo risparmio di tempo può appesantire l'eseguibile: dichiarateli manualmente nel sorgente;

- **API di Windows**: siete cresciuti all'ombra della *VCL*? La *Visual Component Library* è la vera responsabile delle dimensioni, non di rado spropositate, degli eseguibili. Grazie alle chiamate di sistema e a librerie come la *KOL* è possibile ottenere programmi funzionali a partire da 5K.

Il risparmio di memoria durante l'esecuzione è ottenibile applicando le regole dettate dal buonsenso:

Complessità

L'efficienza di un programma può essere caratterizzata dai tempi di esecuzione (efficienza temporale) e/o dalla memoria impiegata (efficienza spaziale). È possibile ottenere una misura dell'efficienza indipendente dalle architetture hardware e software, analizzando la complessità temporale e quella spaziale di un algoritmo. Esistono molteplici notazioni ideate per determinare se un algoritmo è più veloce/lento di un altro, tra le più comuni possiamo annoverare le notazioni asintotiche: O, Θ, Ω.

O GRANDE:
Scrivendo $f(n) \in O(g(n))$ si intende che $f(n)$ è definitivamente maggiorata da $g(n)$. In altre parole esistono un valore k dell'ingresso ed una costante positiva c tali che: $0 \leq f(n) \leq cg(n)$ per ogni $n > k$. È importante notare che per valori di n minori di k potrebbe accadere che $f(n)$ sia maggiore di $cg(n)$, a partire da k risulta però $cg(n) \geq f(n)$!

Nella figura a lato k vale *1*.

OMEGA GRANDE:
Scrivendo $f(n) \in \Omega(g(n))$ si intende che $g(n)$ è definitivamente maggiorata da $f(n)$, ovvero: $\exists\, c,k > 0$ tali che $0 \leq cg(n) \leq f(n)$ per ogni $n > k$.

THETA GRANDE:
Scrivendo $f(n) \in \Theta(g(n))$ si intende che, a partire da $n>k$, $f(n)$ è compresa tra $c1\ g(n)$ e $c2\ g(n)$ dove $c1$ e $c2$ sono due costanti moltiplicative positive, cioè: $\exists\, k, c1, c2$ tali che $c1g(n) \leq f(n) \leq c2g(n)$ per ogni $n > k$.

Di solito si valuta la complessità computazionale di un algoritmo facendo riferimento alla notazione *O grande*. Vediamo un piccolo esempio: vogliamo ordinare 1000 elementi (*n=1000*) e dobbiamo scegliere tra un algoritmo *Bubble-Sort* ed uno *Quick-Sort*. Dopo "ore" spese in considerazioni formali determiniamo che il primo ha, in generale, una complessità *O(n^2)* quindi risulta *O(1.000.000)*, il secondo ha un caso medio che "costa" solo *O(n log n)* e dunque *O(10.000)*. Un bel risparmio di tempo! Se vi è venuta voglia di approfondire l'argomento vi consiglio di consultare un qualsiasi testo accademico di informatica di base.

- Non creiamo automaticamente tutte le form previste dall'applicazione, ma solo quelle strettamente necessarie. Le altre verranno create e distrutte via codice;
- Rinunciamo ai fronzoli e agli abbellimenti grafici eccessivi concentrando l'attenzione sulle caratteristiche fondamentali;
- Impieghiamo al meglio la gestione delle eccezioni, in particolare i blocchi *try…finally*.

L'ultimo passo consiste nel verificare la presenza dei temuti *memory leaks*, ovvero di sprechi di memoria dovuti a errori di allocazione e deallocazione. Per fortuna esistono delle suite di strumenti, più o meno sofisticate e costose, che svolgono gran parte del lavoro al posto nostro. *Memproof* è un tool gratuito che ci permette di scovare i bug più subdoli: gli sprechi di memoria. Gli eseguibili processati da *Memproof* devono essere compilati con le informazioni di debug, si faccia riferimento alla sezione *Linker* delle opzioni di progetto oppure allo *switch –v* del compilatore da linea di comando.

Fig. 5: MemProof segnala eventuali sprechi di memoria (ed altri errori comuni).

Dopo aver caricato l'eseguibile da testare in *MemProof* si avvia l'analisi con il tasto *F9*, ad analisi terminata si ha un resoconto approfondito con tanto di descrizione del problema (Fig. 5).

CONCLUSIONI

Nello spazio che mi è concesso ho tentato di mettere molta carne sul fuoco in modo da stuzzicare la vostra curiosità. L'ottimizzazione è un argomento interessante ma al tempo stesso tra i più complessi, spero di avervi invogliato a testare i vostri programmi con gli strumenti descritti, potreste avere delle sorprese! La programmazione parallela, l'ottimizzazione degli applicativi database e client/server, il nuovo Delphi 7 ed il futuro Delphi .NET meriterebbero intere serie di articoli…
È per questo che vi consiglio di non perdere i prossimi numeri!

Salvatore Meschini

Sistema

Ottimizzare
Programmi Delphi

Complessità
Nella tabella di fianco troviamo una breve disamina sulla complessità e sulla misura dell'efficienza.

Ottimizzare
PROGRAMMI DELPHI

Ambienti di sviluppo sempre più complessi e computer di ultima generazione dalle prestazioni strabilianti sembrano aver reso inutile l'ottimizzazione del codice. Vediamo come, quando e perché non abbassare la guardia.

Sono in molti a sostenere, non a sproposito, che l'ottimizzazione sia un'arte. In effetti scrivere del "buon codice" significa anche sapere amalgamare capacità tecniche e creatività, senza dimenticare una generosa dose di passione.
Il know-how necessario per ottimizzare, in modo adeguato, un programma risulta spesso fuori dalla portata del programmatore alle prime armi. E' necessario infatti conoscere a fondo: le caratteristiche dell'architettura hardware, del sistema operativo, del compilatore e del linguaggio di programmazione.
In questo articolo, rivolto principalmente a programmatori che si avvicinano per la prima volta all'argomento, analizzeremo metodi, strumenti e trucchi utilizzabili per aumentare l'efficienza del codice prodotto ed evitare gli errori più grossolani.

CONTRO E PRO

Può sembrare strano, ma non è facile giustificare l'ottimizzazione, in quanto aumentare le prestazioni di un programma costa in termini di:

- **Tempo:** cercare soluzioni migliori significa impiegare del tempo prezioso, è perciò molto improbabile che il codice ottimale si concili con il time-to-market;
- **Stabilità:** i trucchi usati per raggiungere certe prestazioni possono minare il corretto funzionamento del programma;
- **Portabilità e manutenibilità:** l'impiego di alcune ottimizzazioni lega il codice ad un numero relativamente ristretto di architetture e/o compilatori. Inoltre, in assenza di commenti si rischia di ottenere "spaghetti code", ossia sorgenti intricati, poco comprensibili e difficilmente mantenibili.

A volte però il conseguimento di prestazioni elevate rientra tra i requisiti del progetto ed in ogni caso l'efficienza è una discriminante non indifferente: uno dei fattori chiave nella scelta di un programma rispetto ad un altro similare. Considerate poi che nei dispositivi mobili, di strategica importanza per il futuro, le risorse sono limitate e la razionalizzazione del software diventa una necessità imprescindibile.
La "qualità del codice" è direttamente proporzionale all'esperienza del programmatore, la conoscenza delle tecniche di ottimizzazione rappresenta dunque un influente biglietto da visita.
Nel prosieguo dell'articolo verranno illustrate le regole più semplici e generali per ridurre la dimensione dell'eseguibile, dello spazio occupato in memoria e dei tempi di calcolo.

DA DOVE PARTIRE

In generale per ottimizzare un programma è necessario operare (almeno) a due livelli di astrazione:

- **Livello algoritmo/strutture dati:** lo studio della complessità computazionale fornisce una misura oggettiva della "bontà" di un algoritmo e aiuta il programmatore nella scelta del metodo più idoneo alla risoluzione di un problema. E' bene non dimenticare che ogni struttura dati ha delle peculiarità che potrebbero viziare in maniera determinante le prestazioni;
- **Livello implementazione:** solo dopo aver esaminato in modo esaustivo il livello precedente si può pensare di "mettere mano al codice" e cercare i costrutti, le funzioni di libreria ed i trucchi del mestiere che garantiscono la maggiore efficienza.

Entrambi i livelli prevedono una casistica sterminata e come spesso accade i punti di vista sono i più disparati, in alcuni casi nettamente contrapposti, mi limiterò dunque a fornire tecniche applicabili in generale.
Per motivi di spazio trascureremo argomenti importanti come la programmazione parallela (multithreading, presenza di GPU, …) e l'uso delle proprietà dei dispositivi hardware (branch prediction, pipelining, vincoli di accesso alla memoria cache, …). Spero che una volta letto l'articolo termini quali profiler e memory leak risultino meno astrusi.

Sistema

File sul CD
\soft\codice
\Ottimizzazione Delphi\

Eseguibili

Per ottenere eseguibili dalle dimensioni ridotte bisogna rinunciare alla VCL e scrivere programmi procedurali basati sulla API di Windows. In alternativa si può utilizzare la libreria open-source KOL/MCK prelevabile da http://xcl.cjb.net Sul CD è presente *InAndOut*, un programma di soli 6K in grado di posizionarsi nella *traybar* ed aprire/chiudere il lettore CD-ROM.

Sistema

Ottimizzare
Programmi Delphi

Assert

Durante la fase di debugging/ottimizzazione di un algoritmo è possibile ricorrere alle asserzioni. La procedura *Assert*, descritta nella documentazione di Delphi, valuta un'espressione booleana e nel caso in cui risulti falsa genera un'eccezione di tipo *EAssertionFailed*. Con *Assert* si possono scovare con maggiore facilità bug subdoli: usatela senza parsimonia in fase di sviluppo!

EFFICIENZA TEMPORALE

Anche se i programmatori Delphi fanno ampio ricorso a componenti scritti da terze parti, prima o poi sorge, per chiunque, l'esigenza di scontrarsi con l'implementazione di routine a "basso livello".

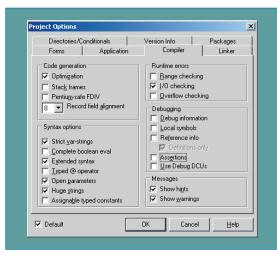
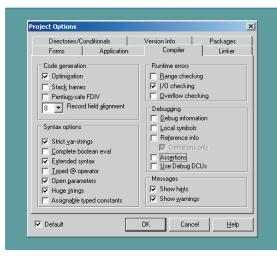



Fig. 1 **Queste impostazioni "suggeriscono" al compilatore di ottimizzare al meglio il codice.**

Come tutti i compilatori moderni Delphi prevede una lunga serie di ottimizzazioni automatiche (Fig. 1), vedremo come aiutarlo nel produrre codice performante (Fig. 2).

Fig. 2: **Con OllyDbg i curiosi possono studiare il codice prodotto dai compilatori. Nell'immagine un ciclo FOR.**

Tra le ottimizzazioni automatiche troviamo:

- **Eliminazione del codice:** il compilatore scarta tutte le istruzioni inutili o comunque non raggiungibili. Le istruzioni considerate utili vengono marcate con un pallino dopo la compilazione, le altre sono completamente ignorate. Per esempio nessuna delle istruzioni della procedura seguente comparirà nell'eseguibile:

```
Procedure Contengo_Solo_Codice_Inutile;
var i,k:Integer;
const c = 0;
begin
 if c = 1 then for i:=1 to 100 do k:=k+1;
                              // Istruzione inutile (c=0)
 i:=0; // Istruzione inutile
 k:=0; // Istruzione inutile
end;
```

- **Variabili nei registri:** esaminando il "tempo di vita" delle variabili Delphi è in grado di memorizzare, senza bisogno di direttive esplicite, le variabili nei registri della CPU. In questo modo vengono drasticamente ridotti i tempi di accesso ed il numero di istruzioni necessarie. Per sfruttare al massimo i registri conviene dichiarare variabili locali e progettare procedure o funzioni con un numero minimo di parametri.

- **Gestione avanzata dello stack:** scrivere funzioni con al massimo tre parametri evita inoltre la creazione di uno stack frame. Le procedure annidate, note anche come *procedure locali*, hanno un impatto negativo sulle prestazioni per via della gestione dello stack: meglio farne a meno.

```
// TODO: mai assegnare un progetto ad un
          programmatore che scrive cose del genere!
Procedure Scarse_Prestazioni (var V:array of char;
                  K:integer; var C: char; var R: Real);
var i:Integer;
Function Non_Dovrei_Essere_Qui (A,B,C:Integer;
                                X,Y,Z: Real) : Real;
begin
 Non_Dovrei_Essere_Qui :=(A+B+C)/(X+Y+Z);
end;

begin
 For i:= 1 TO K DO
  V[i]:=C;
 R:=Random;
end;
```

- **Variabili loop induction**: eliminazione delle *common subexpression, loop unrolling, function inlining*, etc. Per avere una descrizione dettagliata, corredata da esempi, delle tecniche elencate fate riferimento al box *Sul Web*.

Se volete migliorare le vostre applicazioni con il minimo sforzo tenete in considerazione che:

- E' bene ridurre il numero di finestre create automaticamente: oltre a rallentare la fase di avvio consumano molta memoria. Nel pannello *Forms* delle opzioni di progetto (Fig. 3) è opportuno spostare, compatibilmente con le esigenze del programma, le form non necessarie nella lista *Available Forms*. Queste ultime, per non incorrere in violazioni di accesso o sprechi di memoria, verranno create e distrutte dinamicamente solo al momento del bisogno:

```
procedure TMiaFinestra.CreaDinamicamente;
var
  MiaFinestra: TMiaForm;
begin
  MiaFinestra := TMiaForm.Create(Application);
  try
    MiaFinestra.ShowModal; // Mostra la finestra modale
  finally
    MiaFinestra.Free; // Libera le risorse
   end;
end;
```

Fig. 3: **Le risorse vanno allocate solo quando servono, gli utenti vi ringrazieranno!**

- I parametri di tipo strutturato o stringa che non vengono modificati all'interno di una procedura /funzione dovrebbero sempre essere preceduti dalla parola chiave *const*, come dimostra l'esempio incluso sul CD, l'incremento delle prestazioni è ragguardevole. Un discorso analogo vale per gli array dinamici o aperti;

```
//Notate la presenza della parola chiave const!
function Scansione_Stringa(const s: string; const c:
                                          char): integer;
var
  i: integer;
begin
  Result := 0;
  // Esamina la stringa s
  for i := 1 to Length(s) do begin
  // Alla prima occorrenza del carattere c esce dal loop
    if s[i] = c then begin
      Result := i;
      break;
    end;
  end;
end;
```

- Le istruzioni di controllo del flusso *abort*, *break*, *continue* e *exit*, nonostante contribuiscano al peggioramento dello stile di programmazione, possono tornare utili per ottimizzare un algoritmo;

- È importante scegliere oculatamente la migliore combinazione tra prestazioni e precisione quando si opera con tipi reali. Come regola generale si dovrebbe evitare l'utilizzo di tipi reali differenti nella stessa istruzione. L'esperienza dimostra che, se possibile, conviene usare *Round* al posto di *Trunc;*

- Per dare una mano al compilatore le selezioni vanno organizzate in maniera intelligente: date la priorità alla condizione più probabile, non dimenticate l'esistenza del costrutto *case* e raggruppate i rami del *case* per valori vicini;

- Le operazioni di *I/O* compiute su dati residenti in memorie di massa devono essere gestite attraverso buffer al fine di minimizzare gli accessi. Una soluzione "sporca" consiste nel caricamento integrale del file in memoria centrale:

```
Procedure Copia_File (const Sorgente,Destinazione:String);
var
  MioFile2: TMemoryStream;
  Buffer: String;
  MioFile1:TFileStream;
begin
   MioFile1:= TFileStream.Create(Sorgente,
                  fmOpenRead); // Apre il file sorgente
  try
    MioFile1.Seek(0, soFromBeginning);
    MioFile2 := TMemoryStream.Create;
    try
      MioFile2.CopyFrom(MioFile1, 0);
      // Letture e scritture avvengono direttamente
                                            nella RAM
      MioFile2.Read(Buffer, SizeOf(Buffer));
      MioFile2.Write(Buffer, SizeOf(Buffer));
      // Per programmi non "didattici" si può usare il
                                   metodo LoadFromFile
    finally
      MioFile2.SaveToFile(Destinazione); // Salva il file
                                              su disco
      MioFile2.Free;
    end;
  finally
    MioFile1.Free;
  end;
end;
```

- Quando le prestazioni sono davvero importanti si possono realizzare routine in linguaggio Assembly ed integrarle nell'applicazione oppure accontentarsi dell'inline assembler. Non è però sempre vero che routine scritte in Assembly risultino più veloci di quelle implementate in Object Pascal, a meno che non sia assolutamente indispensabile state alla larga dalle complicazioni della programmazione a basso livello.

Sistema

Ottimizzare
Programmi Delphi

Bottleneck

Letteralmente colli di bottiglia, rappresentano le porzioni di codice che richiedono la maggiore quantità di risorse di calcolo. È fondamentale concentrare i nostri sforzi sull'eliminazione, anche solo parziale, dei colli di bottiglia.

Ovviamente non ha senso ottimizzare ogni singola linea di codice, ecco perché sono disponibili degli strumenti che misurano i tempi di esecuzione e forniscono statistiche dettagliate sui colli di bottiglia: i *profiler*. Oltre ai numerosi prodotti commerciali esistono anche soluzioni open-source, per i nostri scopi adotteremo il profiler *GpProfile*. Il funzionamento di base è molto semplice: *GpProfile* inserisce delle istruzioni nel vostro progetto, genera un file contenente il resoconto delle misurazioni e al termine dell'esecuzione lo interpreta. I passi da compiere sono essenzialmente quelli riportati di seguito:

1) Aggiungiamo *GpProfile* al menu *Tools* dell'ambiente Delphi;

2) Dopo aver fatto una copia di sicurezza del progetto dobbiamo "instrumentare" il codice con la combinazione di tasti *Ctrl+I*;

3) A questo punto è sufficiente scegliere le routine da sottoporre al cronometraggio, eseguire il programma ed analizzare i risultati presentati da *GpProfile* (Fig. 4);

Fig. 4: L'output di GpProfile al termine dell'analisi, non ignorate le percentuali!

4) Non dimentichiamo di eliminare le istruzioni inserite dal profiler al punto 2 premendo contemporaneamente *Ctrl* e *F2*.

A questo punto possiamo ricavare delle informazioni preziose sulle porzioni di codice che più di altre contribuiscono al degrado delle prestazioni e quindi richiedono un'implementazione ottimizzata.

EFFICIENZA SPAZIALE

Se il nostro obiettivo è ridurre lo spazio occupato, sia in fase di esecuzione sia in termini di dimensioni dell'eseguibile, possiamo utilizzare una serie di metodi più o meno immediati. Distinguiamo innanzitutto le ottimizzazioni delle dimensioni da quelle relative all'occupazione della memoria.
Per le prime possiamo affidarci a:

- **Package**: i package sono normali librerie condivise DLL caratterizzate però dall'estensione *.bpl*. Compilando un progetto con l'opzione *Build with runtime packages* otteniamo un eseguibile dalle dimensioni minime.
  Lo svantaggio principale è che bisogna distribuire anche le librerie impiegate, di conseguenza i package sono realmente utili solo quando vengono condivisi da diversi progetti. Se i package standard venissero forniti con le varie versioni di Windows, gli eseguibili prodotti da Delphi non avrebbero nulla da invidiare a quelli di altri compilatori;

- **Compressori**: se per voi è importante il numero di byte occupati su disco dal programma allora la soluzione migliore è rappresentata dai compressori di eseguibili. Sul mercato ne troviamo per tutti i gusti, tra i più gettonati segnaliamo l'open-source UPX. L'unico problema dei compressori risiede nella gestione della memoria: i programmi compressi generalmente non sfruttano l'allocazione *on demand* ma vengono decompressi completamente nella RAM. Una strada alternativa è rappresentata dalle utility che rimuovono le sezioni superflue, sul CD allegato alla rivista trovate *StripReloc*;

- **Razionalizzazione delle risorse**: l'eseguibile dovrebbe contenere solo le risorse essenziali, eventuali risorse condivise vanno inserite in DLL appositamente create. È buona norma diminuire il numero di colori delle immagini, quasi sempre ne sono sufficienti 256, e caricarle manualmente senza impiegare le scorciatoie messe a disposizione dall'IDE;

- **Clausole unit**: per agevolare lo smart-linking non dovremmo includere unit inutili, inoltre a meno che non ci siano dipendenze mutue, la loro dichiarazione deve avvenire nella *uses* della sezione *implementation* e non in quella della sezione *interface*;

- **Componenti invisibili**: non cedete alla tentazione di effettuare il drag & drop di componenti non visibili in fase di progettazione, l'esiguo risparmio di tempo può appesantire l'eseguibile: dichiarateli manualmente nel sorgente;

- **API di Windows**: siete cresciuti all'ombra della *VCL*? La *Visual Component Library* è la vera responsabile delle dimensioni, non di rado spropositate, degli eseguibili. Grazie alle chiamate di sistema e a librerie come la *KOL* è possibile ottenere programmi funzionali a partire da 5K.

Il risparmio di memoria durante l'esecuzione è ottenibile applicando le regole dettate dal buonsenso:

VCL

La Visual Component Library è la vera responsabile delle dimensioni, non di rado spropositate, degli eseguibili. Grazie alle chiamate di sistema e a librerie come la KOL è possibile ottenere programmi funzionali a partire da 5K.

Sistema

Ottimizzare
Programmi Delphi

Complessità

L'efficienza di un programma può essere caratterizzata dai tempi di esecuzione (efficienza temporale) e/o dalla memoria impiegata (efficienza spaziale). È possibile ottenere una misura dell'efficienza indipendente dalle architetture hardware e software, analizzando la complessità temporale e quella spaziale di un algoritmo. Esistono molteplici notazioni ideate per determinare se un algoritmo è più veloce/lento di un altro, tra le più comuni possiamo annoverare le notazioni asintotiche: O, Θ, Ω.

O GRANDE:
Scrivendo $f(n) \in O(g(n))$ si intende che $f(n)$ è definitivamente maggiorata da $g(n)$. In altre parole esistono un valore k dell'ingresso ed una costante positiva c tali che: $0 \le f(n) \le cg(n)$ per ogni $n > k$. È importante notare che per valori di n minori di k potrebbe accadere che $f(n)$ sia maggiore di $cg(n)$, a partire da k risulta però $cg(n) \ge f(n)$!

Nella figura a lato k vale *1*.

OMEGA GRANDE:
Scrivendo $f(n) \in \Omega(g(n))$ si intende che $g(n)$ è definitivamente maggiorata da $f(n)$, ovvero: $\exists\, c,k > 0$ tali che $0 \le cg(n) \le f(n)$ per ogni $n > k$.

THETA GRANDE:
Scrivendo $f(n) \in \Theta(g(n))$ si intende che, a partire da $n>k$, $f(n)$ è compresa tra $c1\, g(n)$ e $c2\, g(n)$ dove $c1$ e $c2$ sono due costanti moltiplicative positive, cioè: $\exists\, k, c1, c2$ tali che $c1g(n) \le f(n) \le c2g(n)$ per ogni $n > k$.

Di solito si valuta la complessità computazionale di un algoritmo facendo riferimento alla notazione *O grande*. Vediamo un piccolo esempio: vogliamo ordinare 1000 elementi *(n=1000)* e dobbiamo scegliere tra un algoritmo *Bubble-Sort* ed uno *Quick-Sort*. Dopo "ore" spese in considerazioni formali determiniamo che il primo ha, in generale, una complessità *O(n^2)* quindi risulta *O(1.000.000)*, il secondo ha un caso medio che "costa" solo *O(n log n)* e dunque *O(10.000)*. Un bel risparmio di tempo! Se vi è venuta voglia di approfondire l'argomento vi consiglio di consultare un qualsiasi testo accademico di informatica di base.

- Non creiamo automaticamente tutte le form previste dall'applicazione, ma solo quelle strettamente necessarie. Le altre verranno create e distrutte via codice;
- Rinunciamo ai fronzoli e agli abbellimenti grafici eccessivi concentrando l'attenzione sulle caratteristiche fondamentali;
- Impieghiamo al meglio la gestione delle eccezioni, in particolare i blocchi *try…finally*.

L'ultimo passo consiste nel verificare la presenza dei temuti *memory leaks*, ovvero di sprechi di memoria dovuti a errori di allocazione e deallocazione. Per fortuna esistono delle suite di strumenti, più o meno sofisticate e costose, che svolgono gran parte del lavoro al posto nostro. *Memproof* è un tool gratuito che ci permette di scovare i bug più subdoli: gli sprechi di memoria. Gli eseguibili processati da *Memproof* devono essere compilati con le informazioni di debug, si faccia riferimento alla sezione *Linker* delle opzioni di progetto oppure allo *switch –v* del compilatore da linea di comando.

Fig. 5: **MemProof segnala eventuali sprechi di memoria (ed altri errori comuni).**

Dopo aver caricato l'eseguibile da testare in *MemProof* si avvia l'analisi con il tasto *F9*, ad analisi terminata si ha un resoconto approfondito con tanto di descrizione del problema (Fig. 5).

CONCLUSIONI

Nello spazio che mi è concesso ho tentato di mettere molta carne sul fuoco in modo da stuzzicare la vostra curiosità. L'ottimizzazione è un argomento interessante ma al tempo stesso tra i più complessi, spero di avervi invogliato a testare i vostri programmi con gli strumenti descritti, potreste avere delle sorprese! La programmazione parallela, l'ottimizzazione degli applicativi database e client/server, il nuovo Delphi 7 ed il futuro Delphi .NET meriterebbero intere serie di articoli…
È per questo che vi consiglio di non perdere i prossimi numeri!

Salvatore Meschini

Complessità

Nella tabella di fianco troviamo una breve disamina sulla complessità e sulla misura dell'efficienza.

LEGO

MINDSTORMS ROBOTICS INVENTION SYSTEM 2.0

LEGO

Potrebbe,a prima vista, sembra un classico gioco Lego, in realtà nasconde una vera e propria meraviglia tecnologica, che unisce con estrema semplicità robotica, cibernetica e informatica.

Il kit Lego Mindstorms consente di creare robot dotati di articolazioni, sensori ottici e tattili, ecc. utilizzando i mattoncini Lego e l'RCX, un microprocessore programmabile mediante un kit fornito dalla Lego, o tramite appositi SDK per Visual Basic, C++, ecc.
Utilizzando il software incluso nel kit si puo' scrivere un programma in grado di far muovere i piccoli robot in modo che possano evitare gli ostacoli, seguire un percorso o una fonte luminosa. Il kit (Fig.1) si compone di ben 718 pezzi.
Tra questi troviamo:

- Microcomputer RCX;
- CD-ROM con software RCXCode (sviluppo visuale sul PC);
- Constructopedia: una raccolta di istruzioni per costruire robot di esempio;
- 3 Guided Challenges;
- 6 Pro Challenges;
- Dispositivo di trasmissione ad infrarossi per la comunicazione tra PC e microcontroller RCX
- 718 pezzi LEGO, (2 motori, 2 sensori di contatto, 1 sensore di luminosità).

In aggiunta, è possibile acquistare separatamente altri sensori e motori passo passo. In particolar modo sono disponibili sensori di temperatura, di prossima, di angolazione.
Il progetto lo si deve ad un matematico del MIT *(Massachusetts Institute of Technology)*, promotore,tra l'altro, del LOGO, linguaggio di programmazione diffuso nelle scuole per far apprendere i concetti base della programmazione ai più piccoli.

IL CUORE DEL LEGO MINDSTORMS

Il Lego Mindstorms funziona grazie alla presenza di una sorta di microcomputer capace di coordinare i diversi pezzi che l'utente assembla.
RCX, questo il nome del microcontroller a 8 bit, appositamente realizzato dalla Hitachi, che rappresenta il cervello delle invenzioni di Lego MindStorms.
Tale microcontroller può essere programmato grazie all'interfacciamento ad un comune PC; la connessione avviene meditante un'apposita porta ad infrarosso compresa nella confezione.
Nel kit è altresì presente un CD-Rom contenente un linguaggio visuale per creare gli schemi di funzionamento dell'RCX, il programma, molto semplice ed intuitivo da utilizzare, consente di creare com-

Sul Web

Sito ufficiale LEGO
http://mindstorms.lego.com/eng/default.asp

Lego Shop
http://shop.lego.com

Dave Baum's NQC
http://www.baumfamily.org/nqc/

Fig. 1: La scatola contenente i 718 pezzi del Lego Mindstorms.

Fig. 2: Il cuore del Lego Mindstorms: l'RCX.

plessi schemi di funzionamento, gestibile anche da un ragazzino di 12 anni poco pratico dei linguaggi di sviluppo.
Tuttavia, esistono diversi SDK e kit di sviluppo (Lego Mindstorms SDK, Bricx, NQC – Not Quite C) che permettono allo sviluppatore, più avanzato, di creare vere e proprie applicazioni in linguaggio C, Visual Basic; esistono, in giro per la Rete, anche kit di sviluppo Java, Delphi, C#.

Requisiti tecnici minimi

- **Sistema operativo: Windows 98;**
- **CPU: Pentium II 233 MHz;**
- **RAM: 32 MB;**
- **Spazio disco disponibile su hard disk: 115 MB;**
- **Mouse: Windows compatibile;**
- **Scheda: Sound Blaster 16 Windows compatibile;**
- **CD-ROM: 8X;**
- **Video display: 800 X 600 SVGA con 4 MB RAM, 16 bit di colore.**

IL LINGUAGGIO C PER LO SVILUPPO

Il listato che segue (NQC) mostra come realizzare una semplice applicazione in linguaggio; C si tratta di un'applicazione che consente, al robot mostrato in Fig. 3, di seguire il tracciato di una riga nera segnata su uno sfondo chiaro.

```
#define EYE SENSOR_2
#define LEFT OUT_A
#define RIGHT OUT_C

int tooDark;
int tooBright;

void Setup()
{
    tooBright=EYE;
    PlaySound (SOUND_UP);
    Wait (300);
    tooDark=EYE;
    int delta=(tooBright-tooDark)/3;
    tooDark+=delta;
    tooBright-=delta;
}

task main()
{
    SetSensor (EYE,SENSOR_LIGHT);
    Setup();
            On (LEFT+RIGHT);
    while(true)
    {
        if (EYE<=tooDark)
        {
            Off(LEFT);
            On(RIGHT);
        }
        else if (EYE>=tooBright)
        {
            Off(RIGHT);
            On(LEFT);
        }
        else
        {
            On(LEFT+RIGHT);
        }
    }
}
```

Santo Serra

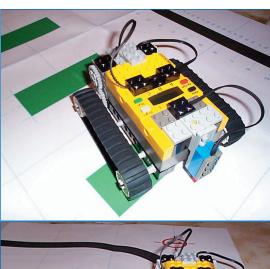
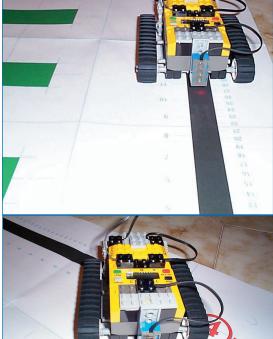



Fig. 3: **Un semplice robot, dotato di sensore di luminosità. Notate come il robot segue, automaticamente, il tracciato nero.**

LEGO

Lego
Mindstorms Robotics Invention System 2.0

Biblioteca

- **DEFINITIVE GUIDE TO LEGO MINDSTORMS II Edition**
Dave Baum's

(Apress)
ISBN: 1-59050-063-5
Prezzo: US $29.99

Si tratta di un testo, in lingua inglese, che consente di seguire passo passo la realizzazione di ben 14 robot. Scritto dal creatore del linguaggio NQC, da la possibilità, a chiunque ne abbia voglia, di realizzare il proprio Robot, analizzandone ogni aspetto, finanche le leggi fisiche che ne regolano il comportamento. Ogni progetto realizzato è ben illustrato e documentato.
Per ognuno dei robot presentati, viene realizzato il codice di funzionamento, sia in linguaggio C, sia mediante l'apposito kit visuale fornito insieme al Lego Mindstorms.
Un testo che non può mancare nella biblioteca di ogni appassionato dei Lego Mindstorms.

Un Software
DI CONTROLLO PER MOTORI PASSO-PASSO

Motori passo-passo

Nello scorso numero è stato proposto uno schema elettronico adatto alla gestione di motori passo passo unipolari: in queste pagine proponiamo il relativo software di controllo, scritto con il linguaggio di programmazione Delphi. Realizzando la semplice scheda hardware proposta in precedenza ed utilizzando il programma riportato in questa sede, si ottiene un sistema semplice ed affidabile per la gestione di motori passo passo.

Si potrebbe dire che l'uomo, nel trascorrere della sua storia, abbia sempre cercato un modo per facilitare la propria esistenza. In epoca moderna, costruendo macchinari che svolgono il lavoro in modo autonomo, l'essere umano ha avuto bisogno di svariati sistemi di propulsione ed azionamento delle proprie apparecchiature che sono divenute sempre più complesse con il passare del tempo. La semplice propulsione non è sufficiente però senza una qualche forma di controllo: inoltre quanto più l'algoritmo di gestione è 'intelligente' e maggiore è la flessibilità dell'apparecchiatura. Ritornando ai nostri motori passo passo, possiamo sicuramente affermare che il circuito proposto in precedenza non è sufficiente al controllo del motore, senza un idoneo programma che ne gestisca le caratteristiche. Abbiamo visto inoltre che questo tipo di motori possono essere azionati in diversi modi, utilizzando diverse sequenze di commutazione delle linee, ottenendo prestazioni diverse che si adattano a differenti applicazioni. In queste pagine vogliamo proporre un programma che sia in grado di gestire un motore passo passo, mediante lo schema elettrico proposto in precedenza, dando modo al lettore di potere variare senso di rotazione, velocità e modo di funzionamento di questo importante componente elettromeccanico. Si assume che venga utilizzato un sistema dotato di WIN 9X oppure Millennium e che venga utilizzata una porta parallela con impostazioni standard LPT1 (Indirizzi *0378h, 0379h 037Ah*).

Fig. 1: Lanciando il programma *'SpuntoStepperMotor.exe'* incluso nel CD, si ottiene la schermata di figura, sulla quale sono direttamente disponibili tutti i comandi necessari alla gestione del motore.

IL PROGRAMMA DI GESTIONE PER MOTORI PASSO PASSO

Nel CD allegato alla rivista il lettore può trovare il programma di gestione completo, compilato e funzionante, completo di codice sorgete e componenti Delphi per la gestione di motori passo passo. Il programma, come dicevamo è in grado di generare una sequenza di commutazione sulle quattro linee del motore, in modo tale da permetterne la rotazione. In Fig. 1 si riporta una schermata del programma e della sua finestra principale. Appare evidente, fin dal primo sguardo, lo schema elettrico del circuito pilota, adatto ad azionare piccoli motori: nel caso si impieghino componenti che necessitano di una potenza maggiore, andrà modificato tutto lo stadio finale, utilizzando transistor di potenza, dotati di alette di raffreddamento ed altri svariati accorgimenti (configurazioni darlington ad esempio). Sul lato sinistro troviamo i comandi essenziali per la gestione del motore: la prima checkbox *'Stepper Motor Running'* stabilisce, quando selezionata, se il motore deve essere in rotazione oppure no; si è preferito l'utilizzo di questo componente visuale anziché di un pulsante per facilitare il lettore in eventuali modifiche. Di seguito sono disponibili due Radiobutton *'FWD'* e *'REW'*: che definiscono il senso di rotazione del motore, avanti ed indietro appunto. La checkbox che segue, chiamata *'Continuous Monitoring'* indica al programma se deve effettuare il controllo dello stato logico della porta parallela o meno: apre inoltre la finestra del *'Parallel Port Manager'*, che consente di visualizzare istan-

File sul CD
\soft\codice \StepperMotor.zip

One Phase ON

Utilizzando la sequenza di funzionamento *'One phase ON'* si procede ad alimentare il motore commutando una fase alla volta: in questo modo si ottiene una ragionevole dissipazione di potenza quando non sono richiesti elevati valori di coppia del motore.

Motori passo-passo

Un Software di controllo per motori Passo-Passo

taneamente ogni variazione di stato delle linee di ingresso/uscita della porta. Nella casella di Spinedit *'Delay between Steps (Msec)'*, come il lettore potrà immaginare, è possibile selezionare il ritardo espresso in millisecondi che deve intercorrere tra un passo e l'altro: ovviamente il reciproco di questo valore esprime la frequenza di funzionamento misurata in Kilohertz. Nella parte bassa della finestra abbiamo infine tre *RadioButtons* denominati *'Pattern'* che indicano il tipo di sequenza che il motore deve seguire: *One phase ON*, *Two phase ON*, oppure *Half Step*: maggiori dettagli in merito verranno riportati in seguito.

LA STRUTTURA DEL PROGRAMMA

Di seguito si riporta il listato relativo al programma di controllo, scritto in Delphi e comunque facilmente trasportabile in qualunque altro linguaggio di programmazione. Nella prima parte del programma, ed in particolare nella clausola *'USES'* si può notare l'utilizzo dei due componenti cardine del programma per la parte di monitoring della porta, ovvero *'SpuntoLedComponent e UnitPortaParallela'*.

```
//*******************************************//
//                                          //
// TSpuntoStepperMotor Delphi 6 Application Ver 1.0  //
//   Copyright 2002 Luca Spuntoni  Sept. 2002  //
//       spuntosoft@tiscali.it                   //
//                                          //
//*******************************************//
unit SpuntoStepperMotorUnit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
                    Graphics, Controls,Forms,Buttons,
  Dialogs, ExtCtrls, Menus, StdCtrls,
               SpuntoLedComponent, UnitPortaParallela,
  UnitAbout, Spin, Math;
```

Nella dichiarazione dei tipi, oltre alla classe principale del programma, troviamo l'array *'TStepperMotorPattern'*, che dovrà contenere la sequenza scelta dall'utente, mediante la selezione di uno dei radiobuttons *'Pattern'*.

```
type
//*********  StepperMotor related arrays  ********//
  TStepperMotorPattern =  Array[0..7] of Word;
                              // Array of step patterns
```

La classe principale discende ovviamente da *TForm* e viene riportata nella sua integrità per motivi di chiarezza: contiene tutte le procedure utilizzata nel programma per la gestione dei motori. In particolare, nella parte delle dichiarazioni *'Public'* troviamo *StepperMotorSteps*, una variabile del tipo *'TStepperMotorPattern'* deputata a contenere la sequenza di commutazione in uso, oltre alla variabile *'PresentStep'* che ha il compito di memorizzare quale particolare passo della sequenza è attivo, in modo tale da rendere possibile l'eventuale inversione del senso di rotazione del motore.

```
//**************  Main Class  **************//
 TSpuntoStepperMotorForm = class(TForm)
   StepperMotorPanel: TPanel;
   MainMenu1: TMainMenu;
   Files1: TMenuItem;
   Exit1: TMenuItem;
   ParallelManager1: TMenuItem;
   ShowManager1: TMenuItem;
   About1: TMenuItem;
   Label1: TLabel;
   Label2: TLabel;
   StepperMotorAttivoCheckBox: TCheckBox;
   DirectionGroupBox: TGroupBox;
   FWDRadioButton: TRadioButton;
   REWRadioButton: TRadioButton;
   DelayToStepSpinEdit: TSpinEdit;
   DelaytoStepTimer: TTimer;
   ContinuousMonitoringCheckBox: TCheckBox;
   PatternGroupBox: TGroupBox;
   OnePhaseOnRadioButton: TRadioButton;
   TwoPhaseOnRadioButton: TRadioButton;
   HalfStepRadioButton: TRadioButton;
   Image1: TImage;
   Label3: TLabel;
   procedure ShowManager1Click(Sender: TObject);
   procedure Exit1Click(Sender: TObject);
   procedure About1Click(Sender: TObject);
   procedure FormShow(Sender: TObject);
   procedure FormCreate(Sender: TObject);
   procedure StepperMotorControllaIngressi;
   procedure StepFWD;
   procedure StepREW;
   procedure DelaytoStepTimerTimer(Sender: TObject);
   procedure DelayToStepSpinEditChange(
                                        Sender: TObject);
   procedure ContinuousMonitoringCheckBoxClick(
                                        Sender: TObject);
   procedure WritePort(PortAddress, PortData:word);
   procedure SelectMotorPattern;
   procedure OnePhaseOnRadioButtonClick(
                                        Sender: TObject);
 private
   { Private declarations }
 public
   { Public declarations }
 StepperMotorSteps   : TStepperMotorPattern;
                              // Array of present pattern
 PresentStep:Integer;
 end;
```

Tra le costanti dichiarate di seguito nel programma, troviamo la dichiarazione di tre array corrispondenti alla sequenze di commutazione delle fasi del motore: si può notare che gli array in questione sono formati da

Two Phase ON

La sequenza *'Two phase ON'* permette un maggiore valore di coppia, dal momento che vengono alimentate due fasi per volta, a discapito di una elevata dissipazione di potenza.

Half Step

Quando si rende necessaria una elevata precisione di movimento del motore ed una maggiore fluidità di rotazione, viene utilizzata la sequenza *'Half Step'*, che permette di raddoppiare il numero di passi che il motore deve effettuare per compiere una rotazione completa: in questo modo di funzionamento il valore di coppia disponibile non è costante, dal momento che vengono alimentate alternativamente una e due fasi.

otto elementi, anche se soltanto la sequenza 'Half Step' ha otto passi differenti, mentre per le altre due vengono semplicemente ripetuti i primi gruppi di quattro passi. Il lettore avrà senz'altro notato che il numero intero corrispondente a ciascuna posizione del motore corrisponde alla codifica decimale del valore binario che deve essere inviato alla porta parallela: Ad esempio per eseguire il primo passo della sequenza *'One phase ON'* dobbiamo alimentare la fase corrispondente al bit D0 della porta parallela, collegata al relativo stadio di potenza: per ottenere ciò occorre inviare alla porta il valore binario '00000001' corrispondente al valore decimale '1' .

```
 Const
//UNIpolar Connection, One phase ON
OnePhaseOnMotorSteps   : TStepperMotorPattern
                                =(1,2,4,8,1,2,4,8);
//UNIpolar Connection, Two phase ON
TwoPhaseOnMotorSteps   : TStepperMotorPattern
                               =(3,6,12,9,3,6,12,9);
//UNIpolar Connection, Half Step
HalfStepMotorSteps     : TStepperMotorPattern
                               =(1,3,2,6,4,12,8,9);
var
 SpuntoStepperMotorForm: TSpuntoStepperMotorForm;
implementation
{$R *.dfm}
```

Per lanciare la routine di *'monitorizzazione'* della porta parallela, è sufficiente selezionare l'opzione *'Show Monitor'* del menu *'Parallel Manager'*, oppure semplicemente attivare la checkbox *'Continuous Monitoring'* sulla finestra principale. In questo modo si attiva la finestra del *'Parallel Port Manager'* già abbondantemente descritto negli appuntamenti precedenti, e che sta riscuotendo un notevole successo da parte dei lettori, come comprovato dal flusso di e-mail che l'autore riceve continuamente sull'argomento.

Come si può notare dalla figura che segue, si assume che le impostazioni della porta parallela siano quelle standard di LPT1, cioè utilizzanti gli indirizzi fisici

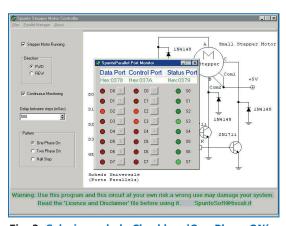



Fig. 2: Selezionando la Checkbox *'One Phase ON'*, si richiede al programma di alimentare il motore in modo tale da attivare una fase alla volta allo scopo di ottenerne la rotazione.

0378h, 0379h e 037Ah. Di seguito si riportano le procedure che provvedono ad attivare il monitor della porta parallela: il loro funzionamento è abbastanza intuitivo, pertanto per motivi di brevità se ne lascia l'analisi al lettore.

```
procedure TSpuntoStepperMotorForm
                    .ShowManager1Click(Sender: TObject);
begin
 ContinuousMonitoringCheckBox.Checked:=True;
 FormPortaParallela.Show;
FormPortaParallela.StepperMotorPortaParallela
                          .SpuntoContinuousMonitoring:=
                ContinuousMonitoringCheckBox.Checked;
end;
procedure TSpuntoStepperMotorForm
                 .ContinuousMonitoringCheckBoxClick(
                                    Sender: TObject);
Begin
FormPortaParallela.StepperMotorPortaParallela
   .SpuntoContinuousMonitoring:=ContinuousMonitoring-
                                   CheckBox.Checked;
If  ContinuousMonitoringCheckBox.Checked Then
   FormPortaparallela.Show else FormPortaparallela.Hide;
end
```

Per la gestione della visualizzazione delle varie finestre di dialogo accessorie del programma e della gestione della chiusura del programma, vengono utilizzate le procedure che seguono, il funzionamento delle stesse è abbastanza ovvio e sul quale non ci dilunghiamo per problemi di spazio.

```
procedure TSpuntoStepperMotorForm.Exit1Click(
                                    Sender: TObject);
begin
SpuntoStepperMotorForm.Close;
end;

procedure TSpuntoStepperMotorForm.About1Click(
                                    Sender: TObject);
begin
SpuntoAbout.ShowModal;
end;
procedure TSpuntoStepperMotorForm.FormShow(
                                    Sender: TObject);
begin
SpuntoAbout.ShowModal;
end;
```

Quando l'utente varia il valore di ritardo nello Spinedit di nome *'DelayToStep'*, viene eseguito l'event handler *'DelayToStepSpinEditChange'* che provvede semplicemente ad impostare il valore del ritardo sul timer *'DelayToStepTimer'*, variando il tempo che intercorre tra due passi consecutivi del motore.

```
procedure TSpuntoStepperMotorForm
           .DelayToStepSpinEditChange(Sender: TObject);
```

Motori passo-passo

Un Software

di controllo per motori Passo-Passo

Monitorizzazione

Per lanciare la routine di *'monitorizzazione'* della porta parallela, è sufficiente selezionare l'opzione *'Show Monitor'* del menu *'Parallel Manager'*, oppure semplicemente attivare la checkbox *'Continuous Monitoring'* sulla finestra principale.

Motori passo-passo

Un Software di controllo per motori Passo-Passo

```
begin DelayToStepTimer.Interval:=
                                        DelaytoStepSpinEdit.Value;
end;
```

All'avvio del programma si rendono necessarie alcune azioni di predisposizione della finestra principale e di inizializzazione delle variabili di controllo del programma. Occorre inoltre selezionare il tipo di sequenza da fare seguire al motore, tramite l'esecuzione della procedura *'SelectMotorPattern'*.

```
procedure TSpuntoStepperMotorForm.FormCreate(
                                        Sender: TObject);
Var
 I:Integer;
 V:Variant;
 S:String;
begin
 //Inizializzazione Variabili di controllo
  StepperMotorAttivoCheckBox.Checked:=False;
  ContinuousMonitoringCheckBox.Checked:=False;
  DelayToStepTimer.Interval:=DelaytoStepSpinEdit.Value;
 //Selezione del tipo di pattern del motore
  SelectMotorPattern;
end;
```

Sistema Operativo

Il programma richiede un sistema operativo WIN 9X oppure Millennium.

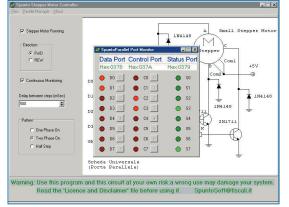



Fig. 3: Selezionando la Checkbox *'Two Phase ON'*, si richiede al programma di alimentare il motore due fasi alla volta, per ottenere il massimo valore di coppia.

Bibliografia

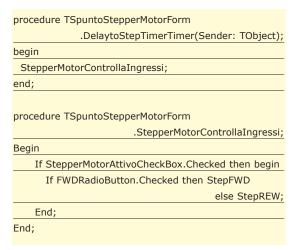
- **CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6 (Luca Spuntoni) ioProgrammo Aprile 2002**

La selezione del tipo di sequenza da utilizzare avviene attraverso la procedura *'SelectMotorPattern'*, che assegna semplicemente all'array *'StepperMotorSteps'* i valori contenenti in una delle costanti *'OnePhaseOnMotorSteps'*, *'TwoPhaseOnMotorSteps'*, oppure *'HalfStepMotorSteps'*.

```
procedure TSpuntoStepperMotorForm.SelectMotorPattern;
Begin
    If OnePhaseOnRadioButton.Checked then
            StepperMotorSteps:=OnePhaseOnMotorSteps;
    If TwoPhaseOnRadioButton.Checked then
            StepperMotorSteps:=TwoPhaseOnMotorSteps;
    If HalfStepRadioButton.Checked then
StepperMotorSteps:=HalfStepMotorSteps;
 End;
```

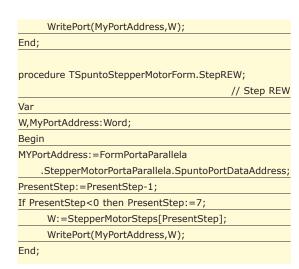
Entrando nel vivo del funzionamento del programma, vediamo come è stato risolto il problema della gestione temporale dell'invio della sequenza di passi alla porta parallela. Il programma comprende un componente *TTimer*, di nome *'DelaytoStepTimer'*; l'event handler di questo componente che si occupa della gestione dell'evento *'OnTimer'* è la procedura *'DelaytoStepTimerTimer'*. In parole più semplici, ogni volta che il timer genera un messaggio *'OnTimer'* la procedura *'DelaytoStepTimerTimer'* viene eseguita. Questa procedura, si preoccupa del controllo dello stato delle uscite della porta parallela, chiamando *'StepperMotorControllaIngressi'* che viene riportata di seguito. Se l'utente ha deciso di fare ruotare il motore in avanti viene eseguito *'StepFWD'* altrimenti viene lanciato *'StepREW'*.

```
procedure TSpuntoStepperMotorForm
                .DelaytoStepTimerTimer(Sender: TObject);
begin
  StepperMotorControllaIngressi;
end;

procedure TSpuntoStepperMotorForm
                              .StepperMotorControllaIngressi;
Begin
      If StepperMotorAttivoCheckBox.Checked then begin
         If FWDRadioButton.Checked then StepFWD
                                             else StepREW;
      End;
End;
```

Le procedure *'StepFWD'* e *'StepREW'* sono molto simili, in breve stabiliscono su quale indirizzo fisico devono inviare la codifica del passo corrente, attraverso il valore contenuto in *'MYPortAddress'*, dopo di ciò provvedono ad incrementare o a decrementare il valore della variabile globale *'PresentStep'* a seconda se il motore debba ruotare in un senso, oppure nell'altro: viene operato a questo punto un controllo per verificare se la sequenza è giunta in uno degli estremi ed in questo caso viene fatta ripartire dall'inizio. Con questo sistema la variabile *'PresentStep'* avanza od indietreggia tra un valore compreso tra *0* e *7*, numero che ci permette a questo punto di ricavare il valore da inviare alla porta da uno degli array contenente le sequenze di passi. Finalmente viene inviato in uscita il valore corrispondente alla codifica binaria della sequenza di attivazione delle fasi del motore attraverso la procedura *'WritePort'*.

```
procedure TSpuntoStepperMotorForm.StepFWD; // Step FWD
Var
W,MyPortAddress:Word;
Begin
MYPortAddress:=FormPortaParallela
      .StepperMotorPortaParallela.SpuntoPortDataAddress;
PresentStep:=PresentStep+1;
If PresentStep>7 then PresentStep:=0;
        W:=StepperMotorSteps[PresentStep];
```

```
        WritePort(MyPortAddress,W);
End;

procedure TSpuntoStepperMotorForm.StepREW;
                                              // Step REW
Var
W,MyPortAddress:Word;
Begin
MYPortAddress:=FormPortaParallela
      .StepperMotorPortaParallela.SpuntoPortDataAddress;
PresentStep:=PresentStep-1;
If PresentStep<0 then PresentStep:=7;
        W:=StepperMotorSteps[PresentStep];
        WritePort(MyPortAddress,W);
End;
```

La scrittura del valore numerico sulla porta parallela avviene per mezzo della procedura *'WritePort'*. In questa occasione il linguaggio Assembler ci viene in aiuto, facilitando l'accesso alle risorse hardware del nostro dispositivo parallelo. A causa dell'accesso diretto alle risorse di sistema, è conveniente utilizzare sistemi operativi WIN 9X, oppure Millennium, dal momento che con WIN2000, oppure XP è possibile che il sistema rifiuti l'esecuzione di questa parte di codice, innalzando un'eccezione di *'Privileged Instruction'*.

```
procedure TSpuntoStepperMotorForm.WritePort(
                                  PortAddress, PortData:word);
// Write PortData over PortAddress if Port Writing is enabled
begin
    PortData := (PortData*256)+PortData;
    asm
    Mov ax,PortData
    Mov dx,PortAddress
    Out dx,ax
    end;

 end;
```

Fig. 4: Selezionando la Checkbox *'Half Step'*, si richiede al programma di alimentare il motore con una sequenza tale da raddoppiarne il numero di passi nominali.

mente in funzione della applicazione che si desidera realizzare, tuttavia si vogliono dare alcune indicazioni per ovviare ad alcuni problemi di semplice risoluzione che potrebbero presentarsi al lettore. Nel caso in cui il motore non accenni affatto a muoversi, si provi a verificare se l'albero è libero di ruotare oppure se rimane fisso in una certa posizione: nel primo caso significa che probabilmente il motore non viene alimentato, nel secondo che vi è qualche errore di cablaggio, in ogni caso per risolvere il problema provvediamo a controllare tutte le connessioni. Se il motore si muove ma anziché ruotare vibra, oppure procede a scatti irregolari, significa che la connessione delle fasi del motore non è corretta, quindi occorre verificarne nuovamente i collegamenti. Può verificarsi un eccessivo surriscaldamento dei transistor, nel caso in cui questi siano sottodimensionati rispetto alle caratteristiche del motore: in questo caso provvediamo a sostituirli con finali di potenza maggiore, eventualmente in configurazione *Darlington* e dotiamoli di una aletta dissipatrice: in casi estremi installiamo una ventola di raffreddamento.

Motori passo-passo

Un Software *di controllo per motori Passo-Passo*

Parametri della Porta Parallela

Si assume che nel sistema utilizzato sia disponibile una porta parallela con parametri standard LPT1 (Indirizzi 0378h, 0379h 037Ah).

COLLAUDO DEL SISTEMA

Una volta realizzato il circuito di controllo, provvediamo a collegarlo alla porta parallela del PC, mediante un cavo di prolunga a 25 poli pin to pin, facendo attenzione a verificare che il cavo sia effettivamente di questo tipo. Procediamo ad alimentare il circuito prima di accendere il PC e lanciamo il programma di controllo. Per provare che tutto funzioni come dovrebbe impostiamo un ritardo di 500 mSec tra un passo e l'altro, in modo da verificare il movimento del motore con accuratezza.

RISOLUZIONE DEI PROBLEMI

Come è già stato detto più volte quanto riportato in questa sede riveste una veste generale e di studio, il circuito di controllo infatti andrebbe disegnato apposita-

CONCLUSIONI

In questo speciale dedicato ai motori passo passo abbiamo appreso come realizzare un circuito di controllo per questi motori, abbiamo esaminato le tecniche relative al loro controllo ed abbiamo analizzato un programma completo per la loro gestione. Per motivi di spazio non abbiamo analizzato alcune tecniche avanzate per il controllo della coppia e della velocità di rotazione dei motori. A questo proposito ricordiamo che l'utilizzo delle tecniche, del codice e del progetto elettronico discussi in quest'articolo, deve essere effettuato con estrema cautela, dal momento che l'autore e l'editore non possono essere accreditati di alcuna responsabilità implicita od esplicita conseguente dall'uso e dal funzionamento del software e dell'hardware esposto in questa sede.

Luca Spuntoni
(spuntosoft@tiscalinet.it)

Un server
WEB NEL TASCHINO DELLA GIACCA

Micro Controller

Si chiama Rabbit, un completo sistema a microprocessore, dalle ridotte dimensioni (più piccolo di un pacchetto di sigarette), in grado di "dialogare" in rete, tramite protocollo TCP/IP. Dotato di diverse porte di input/output consente, per esempio, di pilotare tramite la rete Internet qualunque apparecchiatura elettronica.

Un modulo Rabbit è una particolare scheda elettronica, di dimensioni ridottissime (Fig.1), che riesce ad assolvere a svariati compiti. Il sistema ospita un microprocessore Rabbit a 8 bit con clock interno a 22.1Mhz, una memoria Flash da 256 KByte, memoria RAM da 128 KByte, porta Ethernet 10Base-T da 10Mbit, 26 linee di I/O di uso generale, 4 porte seriali sincrone con baud rate sino a 345.600 bps, un orologio Real-Time con datario, 5 timer a 8 bit, 1 timer a 10 bit,un Watchdog timer.
Un modulo Rabbit può essere così inserito in qualunque progetto elettronico e assolvere a diversi compiti, tra cui, forse quello più spettacolare, il telecontrollo; pensate alla possibilità di interfacciare il modulo ad un qualunque apparato elettrico e comandare quest'ultimo, da qualunque parte del pianeta, tramite una semplice connessione Web.
Unico vincolo, il costante interfacciamento del modulo alla connessione Internet (tramite normale cavo RJ45), ma con ADSL questo non dovrebbe essere un problema.

Fig. 1: Il modulo Rabbit RCM2200.

Distribuita da...
Area SX S.r.l.
INFORMATICA & MICROELETTRONICA
http://www.areasx.com
info@areasx.com
Via Luigi Robecchi Brichetti, 13
00154 - Roma
Tel. 06 57.17.26.79
Fax. 06 57.17.26.95

Il modulo Rabbit viene fornito (separatamente) dall'ambiente di sviluppo *Dynamic C 7.26TSE2*, che permette di programmare il modulo a seconda delle proprie esigenze. Esistono diversi modelli di Rabbit, quello di cui ci occuperemo in questo articolo, è l'RCM2200 che tra l'altro rappresenta anche il modello più economico della gamma.
Il modello, come già accennato in precedenza, dispone di una porta Ethernet che consente di interfacciarlo ad una connessione di Rete, permette quindi di gestire una serie di protocolli di rete, tra i quali: http, ftp, telnet, POP, SMTP. Nei core modules Rabbit coesistono diverse caratteristiche tipiche dei microcontroller: memoria flash, basso assorbimento, linee di I/O TTL, ecc. e caratteristiche tipiche dei PC quali: connessione di Rete, gestione dei protocolli TCP/IP, file system, ambiente di programmazione IDE in linguaggio C, multitasking, ecc.
In Italia il prodotto è distribuito da AreaSX (*www.areasx.com*) di Sergio Tanzilli, all'interno del sito troverete tutte le indicazioni utili per acquistare sia i moduli che l'ambiente di sviluppo. AreaSX produce anche una scheda, denominata *demo board SX01* che, interfacciandosi con il modulo *Rabbit RCM2200*, aiuta gli sviluppatori ad effettuare prototipazioni veloci di progetti. La demo board (Fig.2) può essere acquistata indipendentemente dal development kit e dai moduli Rabbit, in versione già montata e collaudata.

Fig. 2: La demo board SX01

I MODULI DELLA SERIE 2000

Il Rabbit 2000 è un microprocessore basato su una architettura a 8 bit, con aritmetica a 16 bit. La serie 2000 dei Core Modules comprende 4 modelli principali, identificati rispettivamente dalle sigle RCM2000, RCM2100, RCM2200, RCM2300. Ciascuno di questi modelli ha diverse versioni.

RCM20x0

All'interno di questa classe si trovano le versioni RCM2000, RCM2010 ed RCM2020

Tutti questi modelli sono equipaggiati con 40 linee I/O, 4 porte seriali, 5 timer,Real Time Clock; non sono dotati invece di porta ethernet. Sono indicati per progetti entry level e low cost, dove l'accesso alla rete non è utilizzato. Le differenze tra le varie versioni risiedono solamente nella velocità del clock (25.8 MHz per le versioni 2000 e 2010, 18.4 MHz per la versione 2020) e nella dotazione di SRAM.

RCM21x0

All'interno di questa classe si trovano le versioni RCM2100, RCM2110, RCM2120 ed RCM2130

Sono Core Module di dimensioni meno contenute rispetto ad altri modelli equivalenti (misurano 89 x 51 x 22 mm) e sono tutti dotati di porta ethernet; le due versioni 2100 e 2110 hanno il connettore RJ45 con led a bordo, mentre i modelli 2120 e 2130 mettono a disposizione solamente i segnali ethernet sui piedini, consentendo il montaggio di una presa RJ-45 esterna.
Sono modelli superati dagli equivalenti della classe RCM2200, che forniscono più o meno la stessa dotazione hardware in uno spazio più contenuto; l'unico vantaggio di questi modelli è la presenza di alcune linee di I/O in più.

RCM22x0

All'interno di questa classe si trovano le versioni RCM2200, RCM2210 ed RCM2250

Questi modelli, molto simili tra loro, sono sicuramente tra i più interessanti della serie, per le ridotte dimensioni e la dotazione hardware veramente notevole. Comprendono infatti, su una scheda di 4x6 cm, 26 porte di I/O, 4 seriali, una porta ethernet, un RTC, linee di espansione (4 indirizzi, 8 dati ed un I/O Read-Write), timer e watchdog. La differenza tra i modelli 2200 e 2210 consiste nel fatto che il primo ha il connettore ethernet ed i led di attività (LNK e ACT) montati a bordo, mentre il secondo ha solamente i segnali ethernet portati sui piedini, consentendoci di montare un nostro connettore e dei led esterni.
Il modello RCM2250 è sostanzialmente identico al 2200 ma ha un superiore quantitativo di memoria sia Flash che SDRAM (512K per entrambe).

RCM23x0

Questa classe comprende il solo modello RCM2300

Questo Core Module ha caratteristiche sicuramente notevoli: nonostante le sue dimensioni estremamente ridotte (misura 4x3 cm) che consentono una alta integrazione, fornisce una dotazione hardware di tutto rispetto, comprendente 29 linee di I/O, 4 porte seriali, 5 timer ad 8 bit, un timer a 10 bit, real time clock, watchdog.
Altra caratteristica di rilievo è quella di essere pin to pin compatibile con i modelli della serie RCM22x0. Questo significa che nel momento in cui sviluppiamo un progetto basato su questo Core Module possiamo, senza modificare l'hardware realizzato, aggiungere la possibilità di avere la porta ethernet, semplicemente sostituendo il Core Module stesso.

I MODULI DELLA SERIE 3000

I modelli della serie 3000 sono solamente 3, identificati rispettivamente dalle sigle RCM3000, RCM3100 ed RCM3200. Sono caratterizzati da dimensioni maggiori rispetto ai modelli serie 2000, ma hanno caratteristiche di tutto rilievo. La prima differenza fondamentale è che il microcontrollor ha un core a 16 bit, rispetto agli 8 del Rabbit 2000; questo gli garantisce tutta una serie di vantaggi, tra cui un notevole aumento di velocità ed un maggiore spazio di memoria indirizzabile. La quantità di periferiche è notevole, si va dalle 6 porte seriali, alle 56 linee di I/O, unitamente alla capacità di essere alimentato da 1.5V a 3.6V, mantenendo comunque la possibilità di ricevere ingressi TTL (da 0 a 5V).

RCM30x0

All'interno di questa classe si trovano le due versioni RCM3000 ed RCM3010

Questi Core Module, oltre alle dotazioni "di serie" ereditate dalla serie 2000 (52 linee di I/O, 6 porte seriali, 10 timer, real time clock, watchdog, linee di espansione), possiede caratteristiche del tutto innovative: una porta seriale ad infrarossi SDLC/HDLC (con IrDA), 4 porte SPI, 4 uscite PWM (Pulse Width Modulation), un decoder in quadratura per encoder ottici.
Di dimensioni leggermente più grandi della maggior parte dei core module serie 2000 (misura infatti 69 _ 47 _ 22 mm), è una scelta validissima per tutti i progetti, anche i più complessi

RCM31x0

Modelli in questa classe sono l'RCM3100 e l'RCM3110

Due core modules adatti per tutti quei progetti in cui non sia richiesta una porta ethernet. Infatti in soli 47_42 mm integrano tutte le periferiche presenti nei modelli della classe RCM30x0, mantenendo con questi anche la compatibilità pin to pin. Questo consente, analogamente al modello RCM2300, di sviluppare applicazioni che non abbiano bisogno della connettività ethernet e poi migrare, semplicemente sostituendo il core module con uno della classe RCM30x0, ad applicazioni che ne facciano uso.
La differenza tra i due modelli risiede solamente nella

Micro Controller

Un server
WEB nel taschino della giacca

Ambiente di sviluppo

Caratteristiche dell'ambiente IDE Dynamic C SE:

- **Un editor per poter scrivere i sorgenti da compilare**
- **Un compilatore ANSI C**
- **Un loader per scaricare il codice compilato sul modulo Rabbit**
- **Un debugger per poter eseguire passo passo i propri programmi**
- **Una raccolta di librerie standard pronte all'uso**
- **Una completa documentazione tecnica (in inglese)**

Micro Controller

Un server WEB nel taschino della giacca

quantità di memoria a bordo (512K di flash ed altrettanta di SDRAM per l'RCM3100, la metà per il 3110).

RCM32x0

Unico modello in questa classe è l'RCM3200

Il Core Module più potente e completo della gamma. Di dimensioni abbastanza contenute (69_47 mm) lavora con un clock a 44 Mhz e possiede una dotazione di memoria di tutto rispetto: 512Kb di Flash e 768Kb di SDRAM (divisa tra programma e dati). Oltre alle dotazioni "standard" della classe 3000 (52 I/O, 6 seriali, SDLC/HDLC, RTC, ecc.) possiede anche il supporto per convertitori MIR/SIR IrDA.
E' il prodotto da utilizzare per i progetti più complessi, dove siano richieste tutta la velocità e la potenza che questo Core Module può offrire; in progetti meno complessi è preferibile sicuramente un modello delle due classi precedenti.

UNA PROVA PRATICA DEL RABBIT RCM2200

Il kit di sviluppo per RCM2200 nella versione estesa si compone delle seguenti parti:

- Un CD contenente l'ambiente di sviluppo della Z-World Dynamic C SE.
- Un programmatore in-circuit da collegare tra una porta seriale RS232 libera del nostro PC ed il modulo Rabbit da programmare.
- Un modulo Rabbit RCM2200.
- Una demo board prodotta dalla Rabbit Semiconductor.
- Una scheda di registrazione del prodotto.
- Un piccolo manuale in lingua inglese.

In questa prima prova pratica facciamo riferimento alla scheda SX01 prodotta da Area SX con cui è possibile effettuare diverse prove di sviluppo. Maggiori informazioni sulla SX01 sono accessibili sul sito *www.areasx.com*
La prima operazione da compiere, consiste nell'installazione dell'ambiente di sviluppo installando il relativo software direttamente dal supporto CD-Rom fornito nel development kit.
Terminata la fase di installazione, appariranno sul nostro desktop alcune icone ognuna corrispondente ad altrettanti collegamenti a parti dell'SDK: L'icona *DC 7.26TSE RFU* è il collegamento al programma Rabbit *Field Utility* ovvero un loader che consente di scaricare un programma già compilato direttamente sul rabbit senza richiedere l'intero ambiente di sviluppo. E' molto utile in fase di produzione o distribuzione dei nostri lavori ma per ora non dovremo utilizzarla.
L'icona *DCSE TCPIP 7.26TSE* è il collegamento al vero e proprio ambiente di sviluppo. L'icona *TCPIP 7.26TSE Docs* è il collegamento a tutta la documentazione disponibili sul CD. Infatti come avrete certamente notato con il kit di sviluppo non viene fornito alcun manuale se non un brevissimo "Getting started". Su CD invece è presente una notevole mole di documentazione, schemi elettrici, applicazioni d'esempio, ecc. sia in formato HTML consultabile con il browser, sia in formato PDF più adatto per ottenere una copia su carta. Tutta la documentazione è disponibile purtroppo solo in lingua inglese.

INSTALLAZIONE DEL MODULO

Prima di avviare l'SDK occorre collegare l'hardware di cui disponiamo. Iniziamo dal programmatore. Colleghiamo il connettore a vaschetta femmina a 9 sulla nostra porta seriale (la stessa che abbiamo specificato durante l'installazione) e disponiamo l'altra estremità sulla nostra scrivania. A questo punto innestiamo sulla doppia fila di connettori 13x2 pin posti sulla scheda SX01, il nostro modulo RCM2200, così come indicato in Fig. 3.

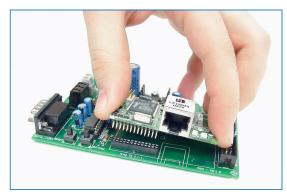



Fig. 3: Inserimento del modulo Rabbit sulla demoboard SX01.

La serigrafia sullo stampato ci aiuterà a posizionare correttamente il modulo. Togliamo lo jumper *JP1* sulla scheda *SX01* situato nei pressi del ponte raddrizzatore *G1* e colleghiamo al morsetto *J4* un alimentatore con tensione compresa tra 9 e 15 volt da almeno *500mA*.
Per quanto riguarda l'alimentatore c'è da sottolineare di badare bene alla corrente erogabile (non inferiore a *500mA*) e la tensione in uscita, ricordiamo ancora una volta compresa tra i 9 e i 15 volt. Non ha importanza se la tensione in uscita sia in continua o alternata o il tipo di spinotto di cui dispone. In tutti i casi dovremo tagliare lo spinotto, spellare i due fili ed inserirli nel morsetto *J4*. Non è necessario prestare alcuna attenzione al verso di inserimento dei due fili grazie alla presenza del ponte raddrizzatore *G1* che penserà a dare alla circuiteria la giusta alimentazione.
Colleghiamo l'alimentatore alla rete elettrica e reinseriamo al suo posto il ponticello JP1 (Fig.4)
Il led rosso power dovrebbe accendersi ad indicare che il circuito è correttamente alimentato.
Inseriamo il connettore contrassegnato con l'etichetta *PROG* sul connettore a 5x2 pin presente direttamente sul modulo Rabbit facendo attenzione che la banda

Demo Board SX01

La demo board SX01 è una scheda progettata e prodotta da Area SX srl. http://www.areasx.com per aiutare gli sviluppatori ad effettuare prototipazioni veloci di progetti basati sui moduli della Rabbit Semiconductor RCM2200.

Fig. 4: Colleghiamo l’alimentatore al circuito.

rossa del cavo flat si trovi in direzione della scritta *J1* presente sullo stampato del Rabbit.
Avviamo quindi l'SDK cliccando sull'icona *DCSE TC-PIP 7.05TSE*. L’applicazione inizierà ad effettuare una scansione delle librerie presenti seguito da un tentativo di comunicazione con il modulo Rabbit. Se appare qualche messaggio di errore, probabilmente abbiamo sbagliato porta seriale, oppure non abbiamo acceso la nostra scheda SX01 o il programmatore non è stato inserito in modo corretto.
Se tutto è stato alimentato seconda la norma, e il programmatore è stato inserito in modo corretto ma nonostante ciò riceviamo ancora un messaggio d’errore, proviamo a cambiare il numero di porta seriale accedendo al menù *Options -> Communications* cambiando nel campo *Port* il numero di porta seriale.
Dal menù *Compile* quindi selezioniamo la voce *Reset Target/Compile BIOS* per richiedere un nuovo tentativo di connessione tra l'SDK ed il modulo Rabbit.

```
Dynamic C Special Edition Version 7.26TSE2 - [Untitled]
File  Edit  Compile  Run  Inspect  Options  Window  Help
Edit  Compile  Assemb  Regs  Stack  ?

// Indirizzo IP e Netmask da assegnare al Rabbit
#define MY_IP_ADDRESS   "192.168.0.103"
#define MY_NETMASK      "255.255.255.0"


// Massima lunghezza per i tipi mime
#define HTTP_MAXNAME    40

// Carica le librerie
#memmap xmem
#use "dcrtcp.lib"
#use "http.lib"

// Importa il file Flash nella memoria del Rabbit
#ximport "rbe\ledcontrol.swf"  ledcontrol_swf

// Prototipi delle funzioni utilizzate
int ledon_cgi(HttpState*);
int ledoff_cgi(HttpState*);

// Set dei tipi di file usati
const HttpType http_types[] = {
 { ".swf", "application/x-shockwave-flash", NULL},
 { ".cgi", "", NULL}
};

// Stringa standard di risposta HTTP
const char httpstr[] =  "HTTP/1.0 200 OK\r\n";

// Associazione tra i nomifile visibili da web
// e le corrispondenti entita' su Rabbit
const HttpSpec http_flashspec[] = {
  { HTTPSPEC_FILE, "/", ledcontrol_swf, NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE, "ledcontrol.swf", ledcontrol_swf, NULL, 0, NULL, NULL},
  { HTTPSPEC_FUNCTION, "ledon.cgi",   0, ledon_cgi, 0, NULL, NULL},
  { HTTPSPEC_FUNCTION, "ledoff.cgi",   0, ledoff_cgi, 0, NULL, NULL}
};

// Inizio programma
main() {

  // Inizializzazione del TCP/IP e dell'HTTP
  sock_init();
  http_init();

  // Assegnazione della porta 80 al server HTTP
  tcp_reserveport(80);

  // Loop principale
  while (1) {

default | Line 71 | Col 1 | Modified | Insert | Edit | COM1: 57600
```

Fig. 5: L’ambiente di sviluppo.

UN PRIMO PROGRAMMA DI TEST

Sulla scheda demoboard SX01 sono presenti due led, uno rosso, l’altro verde,comunemente chiamati led di test. In questa prima applicazione, giusto per verificare il funzionamento dell’intera struttura, scriviamo una semplice applicazione di test, che si limiterà a far lampeggiare alternativamente i due diodi led di test.
Selezioniamo dal menù *File* la voce *New* per creare un nuovo programma da trasferire al Rabbit. Si aprirà una nuova finestra di editing che consentirà di scrivere i nostri programmi in linguaggio C. Digitiamo quindi il seguente sorgente:

```
nodebug void msDelay(unsigned int ams)
{
  auto unsigned long t0;
  for( t0=MS_TIMER; MS_TIMER< ams+t0; ) ;
}
//**********************************
// Inizio programma
//**********************************
main ()
{
  // Imposta la porta D come linea di ingresso/uscita
  WrPortI(PDFR, &PDFRShadow, 0x0);
  // Imposta la porta PD3 come uscita
  BitWrPortI(PDDDR, &PDDDRShadow, 1,3);

// Esegue un loop infinito
  while(1)
  {
    // Attende 500mS
    msDelay(500);
    // Accende il led verde TEST1
    BitWrPortI(PDDR, &PDDRShadow, 1, 3);
    // Spegne il led verde TEST2
    BitWrPortI(PBDR, &PBDRShadow, 0, 7);
    // Attende altri 500mS
    msDelay(500);
    // Spegne il led verde TEST1
    BitWrPortI(PDDR, &PDDRShadow, 0, 3);
    // Accende il led verde TEST2
    BitWrPortI(PBDR, &PBDRShadow, 1, 7);
  }
}
```

Clicchiamo il pulsante *Compile* presente sulla barra degli strumenti.
Selezioniamo dal menu *Run* la voce *Run*. Sulla scheda SX01 dovremo vedere i diodi led *TEST1* e *TEST2*, lampeggiare alternativamente con una frequenza di mezzo secondo.
Per bloccare l'esecuzione del programma selezioniamo la voce *Stop* dal menu *Run*.

CONCLUSIONI

Si conclude qui questa prima panoramica sul modulo Rabbit, nei prossimi appuntamenti approfondiremo il discorso, mostrando come realizzare un’applicazione, e relativa scheda di interfaccia al modulo, per comandare tramite un’applicazione Flash sul Web, una serie di led posti come interfaccia al modulo RCM2200.

Santo Serra

Micro Controller

Un server
WEB nel taschino della giacca

Sul Web

Tutto il materiale descritto in questo articolo è reperibile presso il sito:
http://www.areasx.com

Dal sito è possibile altresì acquistare il modulo RCM2200, la demo board SX01 e il development kit, quest’ultimo per lo sviluppo di applicazioni in linguaggio C.

Non c'é

PASSWORD CHE TENGA!

L'exploit di questo mese riguarda un problema di sicurezza che affligge alcuni modelli di *router ADSL* della Telindus (www.telindus.com) e della Arescom (www.arescom.com), che sembrano gestire le password di accesso con troppa leggerezza.

Premettiamo che non si tratta di un vero e proprio "exploit" di programmazione (non c'è un codice da compilare), ma è comunque interessante studiare questa lacuna di sicurezza, per approfondire le conoscenze sui pacchetti di rete e per esercitare un po' di crittoanalisi. I prodotti affetti dal problema sono la serie *Telindus 112x* e *Arescom NetDSL 1000*: si tratta di router per connessioni ADSL, dotati di un hub/switch interno, accessibili e configurabili via TCP/IP mediante una utility di amministrazione remota proprietaria. L'exploit non è realizzabile da remoto, cioè non può essere usato da un intruso esterno per violare la sicurezza del router, ciò nonostante ritorna molto utile usato dall'interno della propria LAN.

L'OCCORRENTE

L'exploit richiede l'uso dei seguenti software:

1) Packet Sniffer (prelevabile da Internet).

2) Telindus 9100 Maintenance Application (fornita col router o prelevabile da Internet).

Dove e come reperirli? Un packet sniffer è un programma capace di intercettare i pacchetti in transito su una interfaccia di rete; è un software molto utile per la diagnostica, ma, se usato in maniera diversa, è una delle armi più potenti in mano agli hacker. Due ottimi sniffer per Windows 2000/XP sono *Sniff-em* (*www.sniff-em.com*) oppure *Sniffer XP* (*www.ufasoft.com*), che sono sufficientemente attrezzati per i nostri scopi. L'utility di amministrazione *Telindus 9100 M.A.* è invece un programma capace di identificare un router Telindus presente su una LAN e connettersi ad esso per gestirne, mediante interfaccia grafica, la configurazione. Questo programma è in genere fornito col router stesso, ma è anche prelevabile (in via non-ufficiale) al link *http://www.weethet.nl/downloads/Telindus_Local.rar.* Una volta predisposto l'occorrente, può iniziare l'attacco: per prima cosa bisogna eseguire lo sniffer e metterlo in ascolto sull'interfaccia di rete della nostra LAN, quindi lanciare l'applicazione *Telindus 9100 M.A.* e catturare i pacchetti in transito sulla rete. L'indirizzo IP di default assegnato al router è in genere quello standard 192.168.1.1.

ANALISI DI UN PACCHETTO UDP

L'exploit prende spunto da queste considerazioni, rilevabili dall'analisi dei pacchetti catturati dallo sniffer (Fig. 1): appare evidente che l'utility di amministrazione comunica col router usando il protocollo UDP sulla porta 9833; l'utility però non conosce a priori l'indirizzo IP assegnato al dispositivo; l'unica certezza è che router e computer su cui l'utility è in esecuzione si

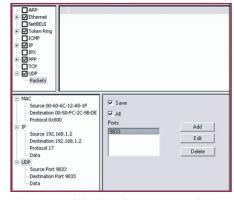



Fig. 1: Lo sniffer in azione, non perde neanche un pacchetto!

trovano sulla stessa subnet e hanno quindi due indirizzi della forma 192.168.1.x. Come fa quindi l'applicazione *Telindus 9100 M.A.* a scovare il router? Ricorre ad un semplice algoritmo di flooding, che prevede l'invio di un pacchetto UDP *(discovery packet)* in broadcast a tutti gli indirizzi IP della rete (192.168.1.255); l'unico dispositivo in grado di reagire a questo pacchetto sarà il router (se presente), che prontamente risponderà al mittente, segnalando la sua presenza. Questa prima sessione di identificazione dura pochi attimi ma è completamente rilevabile seguendo i pacchetti intercettati dallo sniffer, che evidenziano, a questo punto, l'esistenza di un canale di comunicazione tra il router (192.168.1.1) e il computer che sta eseguendo l'utility di amministrazione remota (ad esempio 192.168.1.2).

UN PROTOCOLLO NON PROPRIO SICURO

Superata questa prima fase si noterà un fatto strano nel momento in cui l'utility di amministrazione attende l'immissione della password di accesso al router. Seguendo l'istinto, la prima cosa che ci viene in mente è quella di provare ad entrare con qualche password comune, sperando nell'aiuto della dea bendata, ecco quindi un primo tentativo (a vuoto), un secondo… ma al terzo tentativo notiamo qualcosa di strano! Nessun pacchetto, diretto al router, viene catturato dallo sniffer durante i tentativi di accesso, questo significa una sola cosa: l'utility di amministrazione conosce la password ed esegue la verifica in memoria e non via rete, come ci saremmo aspettati. Quando si tenta l'accesso con una password sbagliata, l'utility di amministrazione dovrebbe trasmettere via rete la pas-

sword al router, attendere la verifica (fatta dal dispositivo) e ricevere la comunicazione dell'esito; tutto ciò in realtà non avviene, perché il programma Telindus 9100 M.A. non trasmette niente al router, quindi conosce a priori la password, che evidentemente gli è stata trasmessa dal router nella sessione iniziale descritta prima. Basta infatti controllare la cronologia dei pacchetti catturati per scoprirne uno insolito, più lungo di tutti gli altri (*>200 bytes*), che una volta analizzato rivela parecchie sorprese…

UN SEMPLICE ATTACCO CRITTOANALITICO

Il pacchetto catturato contiene infatti tutti i dati del router (vedi Fig. 2): nome del dispositivo (evidenziato in verde), serial number, versione del firmware e perfino la password di accesso (evidenziata in rosso)! Una banalità… a cui nessuno aveva pensato, ma che fornisce a chiunque la possibilità di entrare nel router!

```
                          CYPHER-TEXT

0100   00 03 02 00 08 00 00 A2 A3 2B 63 4B 73 23 AB 99    .......¢£+cKs#«
0110   02 0A 22 9A 61 02 93 7B AB A3 2B 90 08 08 00 2B    .."ša."{«£+....
0120   6B 7B AB 9B 28 08 10 01 92 72 22 99 89 91 B1 82    k{«›(...'r"™‰'±
0130   42 29 6A A2 62 49 61 03 B3 2B 91 01 B1 71 81 71    B)j¢bIa.³+'.±q.
0140   91 B9 DA A3 AB 29 02 53 AB 61 01 99 81 01 89 C9    '¹Ú£«).S«a.™..‰
0150   D1 89 B1 D1 99 B1 01 91 81 81 90 09 98 00 10 01    Ñ‰±Ñ™±.'....˜..
0160   E0 08 98 00 30 00 2E C0 9F 0A 88 08 B0 00 30 00    à.˜.0..ÀŸ.ˆ.°.0
0170   85 38 9A 64 0A 00 18 00 10 00 02 00 20 00 10 00    …8šd.........
0180   00 09 30 00 00 09 38 00 00 09 40 00 00 09 80 00    ..0...8...@...€
0190   10 00 10 0A 20 00 00 08 20 00 10 00 00 10 50 00    .... ... .....P
01A0   10 00 00 0A 30 00 10 00 00 0A 48 00 20 00 00 00    ....0.....H. ..
01B0   00 0A 88 00 02 10 28 00 02 11 10 00 00 20 40 00    ..ˆ...(...... @
```

Fig. 2: **I vecchi firmware del router scambiamo pacchetti con la password in chiaro.**

C'è da dire che la Telindus tuttavia, messa a conoscenza del problema, ha subitoapportato alcune modifiche al firmware, che nelle nuove versioni (6.x) usa un algoritmo crittografico per cifrare il pacchetto "incriminato" e per impedire la lettura della password in chiaro rafforzando la sicurezza del router.

Nel pacchetto crittografato la password non è più leggibile. Si può fare qualcosa? Si può solo tentare un attacco critto-analitico al cifrario, sperando che non usi funzioni complesse e confidando nella buona sorte. Confrontiamo il payload di un pacchetto in chiaro (*plain-text*) e di uno cifrato (*cypher - text*) per cercare di capire come sono strutturati (Tab. 1).

I due pacchetti sembrano portare la stessa intestazione (i primi 2+3+2 byte), dopo i quali inizia la sezione informativa: il byte *05* (evidenziato in rosso) nel pacchetto in chiaro indica la lunghezza del campo *<nome router>* ed è seguito infatti dalla stringa *"DSL00"*. Seguono 3 byte (terminazione?) e quindi il byte di lunghezza della password (*0D*, sempre in rosso) con la password in chiaro (*"1111111111111"*). Un attacco di crittoanalisi si basa sulle debolezze del cifrario e sulla conoscenza di porzioni del testo in chiaro: nel nostro caso l'attacco è possibile, perché si conosce il nome del router, visto che viene mostrato (Fig. 4) dall'utility di amministrazione. Nel pacchetto di esempio catturato, il nome del router è *"Telindus ADSL router"*, lungo 20 byte; a partire dal byte *A2* (lunghezza cifrata) inizia il campo *<nome router>*. Non resta che scrivere la tavola crittoanalitica, confrontando i byte criptati con i codici ASCII reali (Tab. 2). Da una prima analisi appare evidente che si tratta di un alfabeto di sostituzioni, un cifrario non difficile da risolvere, in cui ogni lettera è associata ad un codice (si evince osservando le corrispondenze delle lettere *"u"*, *"e"*, *"t"*); tale codice è univoco in qualsiasi punto del testo cifrato. Si osserva inoltre che tutti i codici nell'alfabeto cifrato, terminano con *"B"* o con *"3"*, ad eccezione delle lettere finali di parola. Abbiamo abbastanza materiale per passare all'attacco della password, che inizia subito dopo il terminatore *08 08 00* (Tab. 3). Conosciamo due caratteri della password (*"o"*, *"u"*), sappiamo che è di 5 byte (i terminatori non sono cifrati), ma bisogna ancora scoprire le altre lettere. Costruendo l'alfabeto del cifrario, notiamo che *"r"=93*, *s="9B"*, *"t"=A3*, *"u"=AB* e così via, con incrementi di 8, facendo eccezione per le lettere di fine parola. È facile scoprire il resto della password, ipotizzando *6B="m"* e *9B="s"*; l'ultimo carattere, essendo di fine parola, avrà un codice cifrato diverso da quello standard, ma è comunque facile capire che si tratta di una *"e"* e che la parola chiave è *"mouse"*. La riuscita dell'attacco è legata comunque alla lunghezza del campo *<nome router>*, che deve essere sufficientemente grande da consentire l'attacco crittoanalitico.

Elia Florio

Fig. 3: **Ecco come appare l'utility di amministrazione del router.**

Fig. 4: **Il nome del router viene mostrato dall'utility di manutenzione: è questo l'anello debole della sicurezza.**

Sul Web

Advidsory sulle insicurezze dei router Telindus/Arescom
http://www.securiteam.com/security-news/5DP0A2K7GY.html

Tutorial sull'hackeraggio del router Telindus/Arescom
http://www.weethet.nl/english/adsl_arescomhacked.asp

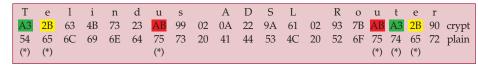
PLAIN-TEXT

0100	00	03	00	01	01	00	00	05 -	44	53	4C	30	30	01	01	00	........DSL00...
0110	0D	31	31	31	31	31	31	31 -	31	31	31	31	31	31	01	02	.1111111111111..

CYPHER-TEXT

0100	00	03	02	00	08	00	00	A2	A3	2B	63	4B	73	23	AB	99	.......¢£+cKs#"™
0110	02	0A	22	9A	61	02	93	7B	AB	A3	2B	90	08	08	00	2B	.."_a."{"£+_...+

Tab. 1: **Confronto fra il payload di un pacchetto in chiaro e di uno cifrato (alcuni byte sono marcatori).**

T	e	l	i	n	d	u	s		A	D	S	L		R	o	u	t	e	r	
A3	2B	63	4B	73	23	AB	99	02	0A	22	9A	61	02	93	7B	AB	A3	2B	90	crypt
54	65	6C	69	6E	64	75	73	20	41	44	53	4C	20	52	6F	75	74	65	72	plain
(*)	(*)					(*)										(*)	(*)	(*)		

Tab. 2: **Tavola crittoanalitica per lo studio del cifrario. Si tratta di un alfabeto di sostituzione.**

Lung. passwd	?	o	u	?	?	Term.
2B	6B	7B	AB	9B	28	08 10 01

Tab. 3: **Conoscendo qualche carattere della password, è facile decifrarla tutta.**

JSP

Procedure
MEMORIZZATE CON JDBC

Nelle applicazioni Web progettate per lavorare da interfaccia verso un database si fa spesso uso di procedure memorizzate. Questa lezione illustra gli scopi ed i funzionamenti delle procedure memorizzate, dimostrando diversi esempi pratici di codice completo.

File sul CD
\soft\codice \codici_jsp13.zip

Le procedure memorizzate sono delle query precompilate. Il loro impiego ha due risvolti positivi: il miglioramento delle prestazioni e la semplificazione del codice. Non tutti i DBMS supportano le procedure memorizzate. Con JDBC è possibile impiegare le procedure memorizzate, purché siano implementate dal gestore di database sfruttato.

L'ESEMPIO GUIDA

Ricollegandoci a quanto già fatto nel corso della precedente lezione, impiegheremo un database di Access come esempio guida. Create un database inizialmente vuoto, quindi inserite al suo interno una tabella, nominata Utenti. La struttura è riportata di seguito:

- **ID**, di tipo *Contatore*, da impiegare come chiave primaria.
- **Nome**, di tipo *Testo*.
- **Cognome**, di tipo *Testo*.
- **Email**, di tipo *Testo*.
- **AnnoNascita**, di tipo *Numerico*.

ID	Nome	Cognome	Email	Anno Nascita
1	Mario	Rossi	rossi@tin.it	1958
2	Luigi	Bianchi	bianchi@libero.it	1972
3	Antonio	Verdi	verdi@tiscali.it	1967
4	Franco	Neri	(NULL)	1948
5	Vittorio	Grigi	(NULL)	1979
6	Massimo	Gialli	(NULL)	1981

Tab. 1: Tabella popolata con dati arbitrari.

Popolate la tabella con qualche record arbitrario Tab: 1. Come al solito, registriamo un DSN di sistema associato alla base di dati. Scegliamo il nome *MioDatabase*.

PREPAREDSTATEMENT

L'interfaccia *PreparedStatement*, del package *java .sql*, estende la già esaminata Statement. Richiamiamo alla memoria le operazioni necessarie per interrogare un DBMS, sfruttando l'interfaccia *Statement*. Per prima cosa, è necessario recuperare un oggetto che implementi tale interfaccia di programmazione. La connessione (un oggetto *Connection* aperto) può fornircelo:

```
Statement statement = connection.createStatement();
```

Ottenuto un oggetto Statement, è davvero semplice interagire con il DBMS, facendo fruttare le regole sintattiche di SQL:

```
ResultSet resultset = statement.executeQuery
                                        (QUERY_SQL);
```

L'impiego di *PreparedStatement* è simile, ma non del tutto identico. Una *PreparedStatement* è una query SQL precompilata. Al momento della creazione dell'oggetto, è già necessario specificare il testo dell'interrogazione che si intende sottomettere al DBMS:

```
PreparedStatement statement =
            connection.prepareStatement(QUERY_SQL);
```

Giacché il testo della query è fornito nel momento stesso in cui l'oggetto *PreparedStatement* è richiesto al gestore della connessione, non è più necessario passare un'istruzione SQL al metodo *executeQuery()*. Per ottenere i risultati ricercati,

Dispense Web del corso

Diversi approfondimenti legati alla lezione odierna e alle precedenti sono reperibili in Internet, tra le dispense Web del corso. L'URL di riferimento è http://www.sauronsoftware.it/dispenseweb/jsp/.

quindi, sarà sufficiente digitare:

```
ResultSet resultset = statement.executeQuery();
```

Impiegando il database di Access allestito in precedenza, verifichiamo le affermazioni con l'esempio completo (*lezione 13_01_01.jsp*) che trovate nello zip presente nel CD.
Quali vantaggi può avere un approccio del genere rispetto a quello esaminato nel corso della lezione precedente? Come anticipato in apertura, è possibile riscontrare due aspetti significativi:

1. L'incremento delle prestazioni.

2. La maggiore semplicità del codice.

Con l'esempio *lezione 13_01.jsp*, ad ogni modo, non è possibile riconoscere immediatamente questi due aspetti. Le procedure memorizzate forniscono reali vantaggi, infatti, quando se ne fa uso frequente e ripetuto, nel corso della stessa pagina JSP. Il metodo *executeQuery()* di una *PreparedStatement* può essere richiamato più di una volta nel medesimo documento JSP. Se una query deve essere eseguita più di una volta, quindi, l'interfaccia *PreparedStatement* permette di non dover digitare due volte la stessa stringa (= maggiore semplicità del codice).
Non solo: le query SQL sottoposte al DBMS mediante l'interfaccia *PreparedStatement* sono precompilate al momento della loro creazione (= incremento delle prestazioni). Nonostante questo, è improbabile che una stessa query debba essere eseguita più di una volta all'interno dello stesso frammento di codice. È più frequente che si debbano eseguire più query simili, ma non del tutto identiche.
Ad esempio, se volessimo prima un elenco di tutti gli utenti nati dopo il 1970 e poi una lista di quelli nati dopo il 1980, dovremmo escogitare ed eseguire due istruzioni assai simili, eppure significativamente differenti:

```
SELECT * FROM Utenti WHERE AnnoNascita >= 1970
SELECT * FROM Utenti WHERE AnnoNascita >= 1980
```

Con JDBC, è possibile eseguire ambo le query attraverso una sola *PreparedStatement* parametrizzata:

```
PreparedStatement statement =
                              connection.prepareStatement(
   "SELECT * FROM Utenti WHERE AnnoNascita >= ?");
```

Osservando questa sintassi, il parametro di selezione collegato all'anno di nascita dell'utente non viene specificato nel momento della precompilazione dell'interrogazione. Sarà necessario aggiungerlo in seguito, prima di eseguire la query, sfruttando i metodi del tipo *setTipoDato()*. Ad esempio, nel nostro caso, potremo agire come segue:

```
statement.setInt(1, 1970);
ResultSet resultset = statement.executeQuery();
// ...
statement.setInt(1, 1980);
resultset = statement.executeQuery();
```

Passiamo in rassegna un esempio completo, che soddisfa la richiesta ipotizzata in precedenza, si tratta del file *lezione 13_02.jsp*.
Specificare i singoli parametri di una *PreparedStatement* è molto semplice. I metodi a disposizione sono tutti del tipo:

```
statement.setTipoDato(numeroParametro,
                                    valoreParametro);
```

Tra i metodi disponibili, è importante annoverare i seguenti:

- *setBoolean(int n, boolean p)*
- *setByte(int n, byte p)*
- *setDate(int n, Date p)*
- *setDouble(int n, double p)*
- *setFloat(int n, float p)*
- *setInt(int n, int p)*
- *setLong(int n, long p)*
- *setShort(int n, short p)*
- *setString(int n, String p)*

Nella documentazione ufficiale di Java potrete trovare maggiori informazioni su questa famiglia di metodi. Più parametri possono essere usati simultaneamente, in una stessa *PreparedStatement*.
Ad esempio:

```
PreparedStatement statement =
   connection.prepareStatement("SELECT * FROM Utenti
             WHERE AnnoNascita BETWEEN ? AND ?");
```

Per ottenere la lista degli utenti nati tra il 1970 ed il 1980, a questo punto, bisognerà usare la sequenza di istruzioni:

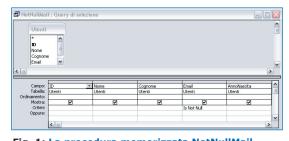



Fig. 1: **La procedura memorizzata NotNullMail.**

JSP

Parametri NULL

Il metodo *setNull()* permette di impostare sul valore nullo un parametro di una procedura memorizzata. Il secondo parametro accettato dal metodo specifica il tipo SQL del campo da verificare. Ogni tipo supportato da JDBC è identificato da un intero. Non è necessario avere una tabella delle corrispondenze, giacché la classe *java.sql.Types* fornisce una serie di appigli mnemonici di più semplice impiego.

JSP

Valori restituiti

Le procedure memorizzate possono restituire dei valori. In tal caso, per poter leggere quanto elaborato, è necessario ricorrere alla sintassi:

```
CallableStatement
    statement = connection.prepareCall(
    "{? = call NomeProcedura(?, ?, ...)}");
```

Maggiori informazioni nella documentazione ufficiale ed in letture apposite.

```
// Imposto il primo parametro.
statement.setInt(1, 1970);
// Imposto il secondo parametro.
statement.setInt(2, 1980);
// Eseguo la query con i parametri correnti.
ResultSet resultset = statement.executeQuery();
```

Il meccanismo è semplice e di facile comprensione.

CALLABLESTATEMENT

Con *PreparedStatement*, come abbiamo appurato, è possibile sfruttare delle istruzioni SQL precompilate, che garantiscono migliori prestazioni e maggiore chiarezza. Le query desiderate sono date in pasto al DBMS al momento della creazione di un oggetto *PreparedStatement*, e possono poi essere ripetutamente sfruttate quante volte si desidera.

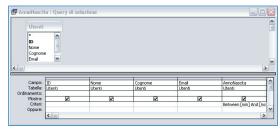



Fig. 2: **La procedura memorizzata AnnoNascita.**

Tuttavia, JDBC ed il DBMS devono ricompilare la medesima istruzione SQL ogni volta che il documento JSP è richiesto da un utente. Ad ogni richiesta, infatti, corrisponde la creazione di un nuovo oggetto *PreparedStatement*, con tutte le conseguenze del caso. Si può fare di meglio, se il DBMS lo permette, ottenendo incrementi ancora più significativi sia nelle prestazioni sia nella semplicità del codice. L'interfaccia *CallableStatement* estende *PreparedStatement*, e permette di richiamare delle procedure memorizzate all'interno del database.

Una query, con *CallableStatement*, non deve essere specificata in linea all'interno del codice della pagina JSP, ma può essere memorizzata perennemente all'interno della base di dati, pronta ad essere sfruttata più e più volte, in quante pagine si desidera. Ogni DBMS impiega tecniche differenti per la preparazione di procedure memorizzate. Sotto Access, ad esempio, è possibile servirsi di una comoda e semplice interfaccia utente.

Riprendiamo il database sfruttato per gli esempi precedenti e dotiamolo di una procedura memorizzata:

1. Aprendo il database con Access, attiviamo la linguetta laterale nominata *"Query"*.
2. Avviamo la procedura *"Crea una query in visualizzazione struttura"*.
3. Selezioniamo la tabella *Utenti* come sorgente di dati.
4. Nell'elenco disponibile, facciamo in modo che tutti i campi della tabella siano selezionati e restituiti (se non siete pratici di Access, fate riferimento alla Fig. 1).
5. Sotto la colonna associata al campo *Email*, alla voce *"Criteri"*, introduciamo la limitazione *"Is Not Null"*.
6. Salviamo la query generata con il nome *NotNullMail*.

Tramite questi passi, è stata memorizzata una query del tipo:

```
SELECT
  ID, Nome, Cognome, Email, AnnoNascita
FROM
  Utenti
WHERE
  Email IS NOT NULL
```

il cui scopo è restituire l'elenco di tutti gli utenti di cui è noto l'indirizzo e-mail. Adesso è possibile richiamare la query *NotNullMail* direttamente da JSP:

```
// Ottengo un oggetto CallableStatement che richiama
// la procedura memorizzata NotNullMail.
CallableStatement statement = connection.prepareCall(
                                   "{call NotNullMail}" );
// Eseguo la query.
ResultSet resultset = statement.executeQuery();
```

Desumerne un esempio funzionante è semplice, come potrete verificare visualizzando il file: *lezione13_03.jsp*. Il codice HTML prodotto in output mostrerà il risultato di Tab. 2.

Nome	Cognome	Email	AnnoNascita
Mario	Rossi	rossi@tin.it	1958
Luigi	Bianchi	bianchi@libero.it	1972
Antonio	Verdi	verdi@tiscali.it	1967

Tab. 2: ***Risultato dell'esecuzione del codice "lezione 13_03.jsp".***

Anche una *CallableStatement* può sfruttare uno o più parametri. Andiamo a dimostrare un esempio di questo tipo.

Tornando ad Access, realizziamo una nuova procedura memorizzata, chiamata *AnnoNascita* (Fig. 2).

Come nel caso precedente, selezioniamo la tabella *Utenti* e trasportiamo tutti i campi disponibili.
Il criterio di restrizione, questa volta, lo applichiamo al campo *AnnoNascita*, con il codice:

```
Between [min] And [max]
```

Questa query, sostanzialmente, corrisponde alla *PreparedStatement* già esaminata in precedenza:

```
SELECT * FROM Utenti WHERE AnnoNascita BETWEEN ? AND ?
```

Potrà essere richiamata alla seguente maniera:

```
CallableStatement statement = connection.prepareCall(
                                  "{call AnnoNascita(?, ?)}");
```

I parametri potranno essere impostati come di consueto:

```
statement.setInt(1, estremoInferiore);
statement.setInt(2, estremoSuperiore);
```

L'intero codice che se ne desume è riportato di seguito:

```
<%@ page import="java.sql.*" %>
<%!
// Nome del driver.
String DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
// Indirizzo del database.
String DB_URL = "jdbc:odbc:MioDatabase";
%>
<html>
<head>
  <title>Corso di JSP, Lezione 13, Esempio 4</title>
</head>
<body>
  <%
  // Carico il driver.
  Class.forName(DRIVER);
  // Preparo il riferimento alla connessione.
  Connection connection = null;
  try {
    // Apro la connesione verso il database.
    connection = DriverManager.getConnection(DB_URL);
    // Ottengo un oggetto CallableStatement che richiama
    // la procedura memorizzata AnnoNascita.
    CallableStatement statement =
     connection.prepareCall( "{call AnnoNascita(?, ?)}" );
    // Specifico i parametri.
    statement.setInt(1, 1970);
    statement.setInt(2, 1980);
    // Eseguo la query.
    ResultSet resultset = statement.executeQuery();
  %>
  <table border="1">
    <tr>
      <td><b>Nome</b></td>
      <td><b>Cognome</b></td>
      <td><b>E-mail</b></td>
      <td><b>AnnoNascita</b></td>
    </tr>
  <%
    // Scorro e mostro i risultati.
    while (resultset.next()) {
      String nome      =  resultset.getString("Nome");
      String cognome  =  resultset.getString("Cogno-
me");
      String email     =  resultset.getString("Email");
      boolean emailIsNull = resultset.wasNull();
      int anno          =  resultset.getInt("AnnoNascita");
    %>
    <tr>
      <td><%= nome %></td>
      <td><%= cognome %></td>
      <td>
        <% if (!emailIsNull) { %>
        <a href="mailto:<%= email %>"><%= email
%></a>
        <% } else { %>
        N.D.
        <% } %>
      </td>
      <td><%= anno %></td>
    </tr>
    <%
    }
    %>
  </table>
  <%
  } catch (SQLException e) {
    // In caso di errore...
  %><b>Eccezione:</b> <%= e.toString() %><%
  } finally {
    if (connection != null) connection.close();
  }
  %>
</body>
</html>
```

L'output restituito comprenderà la Tab. 3.

Nome	Cognome	Email	AnnoNascita
Luigi	Bianchi	bianchi@libero.it	1972
Vittorio	Grigi	(NULL)	1979

Tab. 3: **Output restituito dal codice *"lezione 13_04.jsp"*.**

CONCLUSIONI

Con questa lezione si conclude la panoramica che questo corso dedica all'impiego di JDBC. Eventuali approfondimenti possono essere svolti sfruttando la documentazione ufficiale di Java o dei manuali specifici per JDBC e SQL.

Carlo Pelliccia

Bibliografia

• **JSP, LA GUIDA COMPLETA**
Phil Hanna
(McGraw-Hill)
ISBN 88-386-4207-9
2001

• **IMPARARE JAVASERVER PAGES IN 24 ORE**
Jose Annunziato, Stephanie Fesler Kaminaris
(Tecniche Nuove)
ISBN 88-481-1306-0
2001

• **JAVA, LA GUIDA COMPLETA**
Patrick Naughton & Herbert Schildt
(McGraw-Hill)
ISBN 88-386-0416-9
1997

• **JAVA 2, TUTTO & OLTRE**
Jamie Jaworski
(Apogeo)
ISBN 88-7303-466-7
1999

Visual Basic .NET

Le finestre DI DIALOGO

In questo nuovo appuntamento parleremo delle piccole finestre che appaiono sullo schermo per interagire con l'utente e recuperare informazioni.

Le proprietà di OpenFileDialog

ADDEXTENSION, indica se viene aggiunto automaticamente un'estensione ad un nome di file quando l'utente non la inserisce.
CHECKFILEEXISTS, indica se nella finestra di dialogo viene visualizzato un avviso quando l'utente specifica un nome di file inesistente.
CHECKPATHEXISTS, indica se nella finestra di dialogo viene visualizzato un avviso quando l'utente specifica un percorso inesistente.
DEFAULTEXT, indica l'estensione del file predefinita.
DEREFERENCELINKS, indica se la finestra di dialogo restituisce la posizione del file a cui fa riferimento il collegamento o se restituisce la posizione del collegamento.
FILENAME, contiene il nome del file selezionato nella finestra di dialogo. È una proprietà a sola lettura.
FILENAMES, contiene i nomi di tutti i file selezionati nella finestra di dialogo. È una proprietà a sola lettura.
FILTER, indica la stringa filtro del nome file corrente, che stabilisce le opzioni visualizzate nelle casella relative al tipo di file nella finestra di dialogo.

Le finestre di dialogo forniscono la consueta interfaccia utente presente nelle applicazioni Windows utilizzata per mostrare dei messaggi, oppure per richiedere all'utente dati necessari all'applicazione. VB.NET fornisce molte finestre di dialogo standard che è possibile adattare alle applicazioni, ad esempio la finestra *Apri*, rappresentata dal controllo *OpenFileDialog*, oppure la finestra di *Stampa* rappresentata dal controllo *PrintDialog* e così via. In VB.NET è possibile, inoltre, progettare finestre di dialogo completamente personalizzate in base ad esigenze particolari. Una finestra di dialogo, in pratica, non è altro che una form modale con un insieme di controlli (tipicamente Label, Textbox e Button), la cui proprietà *FormBorderStyle* è impostata su *FixedDialog*.

CREARE UNA FINESTRA DI DIALOGO DA ZERO

Per descrivere come realizzare una finestra di dialogo personalizzata, realizziamo una semplice applicazione d'esempio per il calcolo dell'area di un rettangolo (rileggetevi l'articolo di Ottobre per notare le differenze). Come di consueto creiamo un nuovo progetto, dal nome *CalcoloArea* e modifichiamo il nome di *Form1* in *FormArea*. Nella finestra *FormArea* inseriamo un Button dal nome *ButtonCalcola*. Quando l'utente clicca sul pulsante verrà mostrata una prima finestra di dialogo in cui verrà chiesto di inserire il valore della base del rettangolo, una seconda finestra di dialogo in cui verrà chiesto di inserire il valore dell'altezza del rettangolo, ed infine una terza finestra di dialogo che mostrerà il risultato del calcolo. Per creare la prima finestra di dialogo si deve aggiungere un form al progetto e modificarne alcune proprietà, i passi da seguire saranno:

- Selezionare la voce: *Progetto/Aggiungi Windows Form.*
- Digitare il nome della finestra *(FormDialogoBase)* nella casella di testo *Nome.*
- Cliccare sul pulsante *Apri.*
- Visualizzare la finestra delle proprietà della form in cui modificare le seguenti proprietà per personalizzarne l'aspetto:
- *FormBorderStyle* in *FixedDialog.*
- *MinimizeBox* e *MaximizeBox* su *false,* poiché le finestre di dialogo in genere non includono pulsanti di riduzione ad icona e di ingrandimento.

Le finestre di dialogo vengono visualizzate come finestre modali, e cioè, come finestre a scelta obbligatoria, che impediscono all'utente di eseguire operazioni al di fuori della finestra di dialogo. Per visualizzare una finestra come modale, si deve utilizzare il metodo *ShowDialog* (descritto nei numeri precedenti) pertanto si deve scrivere il codice seguente:

```
Private Sub ButtonCalcola_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
                                        ButtonCalcola.Click
        Dim FBase As New FormBase()
        FBase.ShowDialog()
End Sub
```

La finestra *FormBase* viene utilizzata per richiedere all'utente di inserire i dati relativi alla base del rettangolo, quindi è importante sapere in che modo viene chiusa tale finestra, in altre parole se ha prodotto un risultato. Se ad esempio l'utente chiude la finestra dalla crocetta in alto a destra, i dati inseriti non vengono memorizzati ma eliminati. Per verificare come viene chiusa una finestra di dialogo si può utilizzare la proprietà *DialogResult.* In fase di progettazione è possibile impostare la proprietà *DialogResult* per tutti i controlli *Button* presenti nella finestra di dialogo. Nella finestra *FormBase* inseriamo una *Label,* un *TextBox* e due pulsanti. Visualizziamo la finestra delle proprietà e variamo le proprietà dei controlli appena disegnati:

- Selezionando *Label1* variamo la proprietà *Text* in *Base.*
- Selezionando *TextBox1* variamo la proprietà *Name* in *TextBoxBase* e la proprietà *Text* nella stringa vuota.
- Selezionando *Button1* variamo la proprietà *Name* in *ButtonOk,* la proprietà *Text* in *Ok* e la proprietà *DialogResult* in *OK.*

- Selezionando *Button2* variamo la proprietà *Name* in *ButtonCancella*, la proprietà *Text* in *Cancella* e la proprietà *DialogResult* in *Cancel*.

Dalla finestra (nel nostro caso *FormCalcola*) che visualizza la finestra di dialogo, chiamata anche form padre della finestra di dialogo, è possibile utilizzare il valore della proprietà *DialogResult* per stabilire se è stato scelto *OK* o *Annulla*.
In base alla proprietà *DialogResult* restituita si può decidere se è necessario recuperare o meno le informazioni della finestra di dialogo.

```
If FBase.DialogResult = DialogResult.OK Then
        base = CDbl(FBase.TextBoxBase.Text)
Else
        base = 0
End If
```

In maniera analoga disegniamo la finestra *FormAltezza*. Il risultato dell'operazione di calcolo dell'area del rettangolo, sarà infine visualizzato in una finestra di dialogo predefinita, utilizzando *MessageBox*.

```
Private Sub ButtonCalcola_Click(ByVal sender As
  System.Object, ByVal e As System.EventArgs) Handles
                                    ButtonCalcola.Click
     Dim base As Double
     Dim altezza As Double
     Dim risultato As Double
     Dim FBase As New FormBase()
     FBase.ShowDialog()
     If FBase.DialogResult = DialogResult.OK Then
        base = CDbl(FBase.TextBoxBase.Text)
     Else
        base = 0
     End If

     Dim FAltezza As New FormAltezza()
     FAltezza.ShowDialog()
     If FAltezza.DialogResult = DialogResult.OK Then
        altezza = CDbl(FAltezza.TextBoxAltezza.Text)
     Else
        altezza = 0
     End If

     risultato = base * altezza
     MessageBox.Show(risultato)
  End Sub
```

LA FINESTRA DI DIALOGO MESSAGEBOX

La finestra di dialogo *MessageBox* è senza dubbio quella che vi troverete ad utilizzare più spesso. Questa finestra permette di mostrare messaggi personalizzati all'utente, ed è in grado di accettare scelte tramite uno o più pulsanti. Normalmente è composta da quattro parti:

- La barra del titolo che identifica lo scopo della finestra di dialogo, ad esempio *"Richiesta Conferma"*.
- Il messaggio che appare nella finestra, ad esempio *"Sei sicuro di voler continuare?"*
- Un'icona in grado di attirare l'attenzione, ad esempio l'icona con il punto di domanda.
- Uno o più pulsanti di comando, ad esempio i pulsanti *Si* e *No*.

Per visualizzare la finestra di messaggio, si deve utilizzare il metodo *Show*. Il metodo *Show* può essere richiamato in più modi, riportiamo di seguito la sintassi di quelli più comuni:

```
MessageBox.Show(testo)
MessageBox.Show(testo, etichetta)
MessageBox.Show(testo, etichetta, bottone, icona)
MessageBox.Show(testo, etichetta, bottone, icona,
                                   bottoneDiDefault)
```

In cui:

- **testo** - rappresenta il messaggio che verrà visualizzato all'interno della finestra di dialogo. Può essere una qualsiasi stringa ed è l'unico parametro obbligatorio.
- **etichetta** - rappresenta la stringa visualizzata nella barra del titolo della finestra. Se viene omesso, nella barra del titolo non apparirà nessun testo.
- **bottone** - permette di visualizzare uno o più pulsanti, tra quelli disponibili, nella finestra di dialogo. Se viene omesso, verrà visualizzato il bottone *OK*.
- **icona** - permette di visualizzare un'icona, tra quelle disponibili, nella finestra di dialogo. Se viene omesso, non verrà visualizzata nessun'icona.
- **BottoneDiDefault** - permette di specificare quale pulsante, tra quelli visualizzati, verrà impostato come predefinito. L'uso di questo parametro, consente all'utente di leggere il messaggio e di premere il tasto *Invio* per indicare l'azione del pulsante predefinito.

Analizziamo ora i possibili valori per *bottone*, *icona* e *BottoneDiDefault*. Le icone che si possono visualizzare nella finestra *MessageBox* sono:

- **Asterisk**, **Information -** consentono di visualizzare un'icona contenente un simbolo formato da una lettera i minuscola racchiusa da un fumetto.
- **Error, Hand, Stop** - consentono di visualizzare un'icona contenente un simbolo formato da una *X* bianca racchiusa da un cerchio su sfondo rosso.
- **Exclamation, Warning** - consentono di visualizzare un'icona contenente un simbolo formato da un punto esclamativo racchiuso da un triangolo su sfondo giallo.
- **Question** - consente di visualizzare un'icona contenente un simbolo formato da un punto interrogativo racchiuso da un fumetto.

Visual Basic .NET

Altre proprietà OpenFileDialog

FILTERINDEX, indica l'indice del filtro attualmente selezionato nella finestra di dialogo.
INITIALDIRECTORY, Indica la directory iniziale visualizzata dalla finestra di dialogo.
MULTISELECT, ottiene o imposta un valore che indica se la finestra di dialogo consente la selezione multipla di file.
READONLYCHECKED, ottiene o imposta un valore che indica se è selezionata la casella di controllo di sola lettura.
RESTOREDIRECTORY, indica se la finestra di dialogo ripristina la directory corrente prima della chiusura.
SHOWHELP, indica se viene visualizzato il pulsante di help nella finestra di dialogo.
SHOWREADONLY, indica se la finestra di dialogo contiene una casella di controllo di sola lettura.
TITLE, indica il titolo della finestra di dialogo che viene visualizzato nella barra del titolo.

Visual Basic .NET

Alcune proprietà di ColorDialog

ALLOWFULLOPEN indica se l'utente può utilizzare la finestra di dialogo per definire colori personalizzati.
ANYCOLOR, indica se nella finestra di dialogo sono visualizzati tutti i colori disponibili nel set di colori di base.
COLOR CONSENTE, di ottenere o impostare il colore selezionato dall'utente.
CUSTOMCOLORS, indica il gruppo di colori personalizzati visualizzato nella finestra di dialogo.
FULLOPEN, indica se i controlli utilizzati per creare colori personalizzati sono visualizzati all'apertura della finestra di dialogo.
SOLIDCOLORONLY, indica se nella finestra di dialogo la scelta sarà consentita soltanto ai soli colori in tinta unita

Le combinazioni di pulsanti che si possono visualizzare nella finestra *MessageBox* sono:

- **AbortRetryIgnore** - indica che la finestra di messaggio contiene i pulsanti *Interrompi*, *Riprova* ed *Ignora.*
- **OK -** indica che la finestra di messaggio contiene il pulsante *OK*.
- **OKCancel -** indica che la finestra di messaggio contiene i pulsanti *OK* ed *Annulla*.
- **RetryCancel** - indica che la finestra di messaggio contiene i pulsanti *Riprova* ed *Annulla*.
- **YesNo -** indica che la finestra di messaggio contiene i pulsanti *Sì* e *No*.
- **YesNoCancel -** indica che la finestra di messaggio contiene i pulsanti *Sì*, *No* ed *Annulla*.

Quando l'utente preme uno dei tasti riportati nella finestra di dialogo, viene valorizzata la proprietà *DialogResult* corrispondente. I valori ammessi, per indicare che un pulsante di una finestra *MessageBox* è quello predefinito, sono:

- **DefaultButton1 -** indica che il primo pulsante nella finestra messaggio deve essere il pulsante predefinito.
- **DefaultButton2 -** indica che il secondo pulsante nella finestra messaggio deve essere il pulsante predefinito.
- **DefaultButton3 -** indica che il terzo pulsante nella finestra messaggio deve essere il pulsante predefinito.

Il pulsante predefinito prescelto viene valorizzato in ordine da sinistra verso destra. Se avete visualizzato i pulsanti *AbortRetryIgnore* ed impostate il valore di *BottoneDiDefault* su *DefaultButton3*, allora il pulsante predefinito sarà il pulsante *Ignora*.
Supponiamo ora di aver scritto un'applicazione che salva alcuni dati su database, se l'utente distratto chiude il programma prima di aver salvato effettivamente i dati è bene avvisarlo.
In questo caso possiamo scrivere il codice seguente che visualizza il messaggio di avviso riportato in figura:

```
If MessageBox.Show("Vuoi salvare i dati prima di uscire ?",
        "Richiesta Conferma", MessageBoxButtons.YesNo-
    Cancel, MessageBoxIcon.Question, MessageBoxDefault-
        Button.Button3) = DialogResult.No Then Exit Sub
```

I CONTROLLI FINESTRE DI DIALOGO

VB.NET mette a disposizione alcuni controlli che permettono di visualizzare finestre di dialogo standard, per utilizzarli all'interno della vostra applicazione si deve selezionare il controllo prescelto nella casella degli strumenti e trascinarlo sulla form (al solito essendo un controllo invisibile in fase di esecuzione verrà mostrato nella barra delle componenti).
In particolare descriveremo i seguenti controlli finestre di dialogo standard:

- **OpenFileDialog** - consente di aprire un file.
- **SaveFileDialog** - consente di selezionare un file da salvare e la posizione in cui deve essere salvato.
- **ColorDialog** - consente di selezionare o aggiungere un colore da una tavolozza predefinita.
- **FontDialog** - consente di selezionare un tipo di carattere tra quelli installati nel sistema.
- **PrintDialog** - consente di selezionare una stampante e le pagine da stampare.
- **PageSetupDialog** - consente di impostare i dettagli di una pagina per la stampa.

Per visualizzare una finestra di dialogo standard, si deve utilizzare il metodo *ShowDialog*, così come si è visto in precedenza.
Ad esempio nel caso di un controllo *OpenFileDialog* (dal nome *OpenFileDialog1*) si deve scrivere:

```
OpenFileDialog1.ShowDialog()
```

Anche le finestre di dialogo standard restituiscono la proprietà *DialogResult*. Ad esempio nel caso di un controllo *OpenFileDialog*, la proprietà *DialogResult* restituisce i valori *Ok* o *Cancel* che corrispondono, rispettivamente, ai pulsanti *Apri* e *Annulla* della finestra di dialogo.
L'utilizzo di questi controlli evita la necessità di scrivere codice per implementare il disegno dell'interfaccia, ma resta sempre a carico del programmatore la scrittura del codice.

Il controllo OpenFileDialog

La finestra di dialogo *Apri* permette agli utenti di navigare tra le cartelle del proprio computer locale o di qualsiasi computer in rete e di selezionare un file da aprire. La finestra di dialogo restituisce il percorso completo ed il nome del file selezionato nella finestra.
Il controllo non apre e legge un file, ma è semplicemente un'interfaccia che permette ad un utente di individuare e specificare il file che l'applicazione deve aprire. È possibile utilizzare la proprietà *FilterIndex* per impostare l'estensione dei file che sarà possibile visualizzare, ad esempio soltanto i file testo con estensione *.TXT*. La stringa che definisce il filtro è composta da due elementi separati da una barra verticale (per intenderci il simbolo che si trova di solito di fianco al tasto "1"), l'etichetta che appare nella casella di riepilogo *Tipo File* ed il filtro stesso.
Se ad esempio si vuole permettere la visualizzazione dei file di tipo testo con estensione **.txt*

```
Text files (*.txt)|*.txt
```

Si possono anche definire dei filtri multipli, in questo

caso ogni filtro deve essere separato dalla barra verticale:

```
Files di testo (*.txt)|*.txt|Tutti i files (*.*)|*.*
```

In caso di filtri multipli, per definire il filtro visualizzato di default si deve impostare la proprietà *FilterIndex* ponendola uguale all'indice numerico del filtro desiderato, nel caso precedente si deve impostare *FilterIndex* pari ad 1 oppure a 2. Per stabilire quale file è stato selezionato dall'utente si deve utilizzare la proprietà *FileName*, che memorizza il nome del file preceduto dal percorso completo in cui si trova. Se l'utente clicca sul tasto *Annulla* la proprietà *FileName* sarà pari alla stringa vuota "".

Il controllo SaveFileDialog

La finestra di dialogo *Salva* con *Nome* assomiglia molto alla finestra *Apri*. Le uniche differenze risiedono nelle etichette e nella barra del titolo, dove al posto del testo *apri* viene visualizzato il testo *Salva*. Anche per questo controllo è possibile definire dei filtri sull'estensione del file da salvare con le stesse modalità del controllo *OpenFileDialog*. Le uniche differenze stanno nella presenza di altre proprietà come ad esempio la proprietà *CreatePrompt* o *OverwritePrompt* che poste a *True* visualizzano un messaggio nel caso venga creato un nuovo file o si sovrascriva un file esistente. Naturalmente il controllo non salva il file in automatico.

Il controllo ColorDialog

Il controllo *ColorDialog* permette di visualizzare la finestra di dialogo *Colore* che consente all'utente di selezionare un colore o di crearne di personalizzati. Quando l'utente chiude la casella di dialogo, il colore selezionato viene memorizzato nella proprietà *Color*.

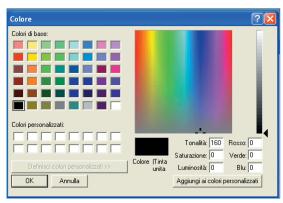



Fig. 1: Finestra di dialogo per la selezione di un colore.

Ad esempio si può assegnare il colore selezionato alla proprietà *ForeColor* o *BackColor* di un altro controllo (ad esempio un *TextBox*)

```
TextBox1.ForeColor = ColorDialog1.Color
```

La matrice CustomColor contiene tutti i colori personalizzati definiti dall'utente.

Il controllo FontDialog

Il controllo *FontDialog* permette di visualizzare la finestra di dialogo *Carattere* che consente all'utente di selezionare diversi caratteri, stili e dimensioni. Ogni volta che l'utente seleziona un'opzione, questa finestra visualizza una casella di anteprima con il carattere corrispondente alle scelte effettuate.
Per recuperare le scelte effettuate dall'utente il controllo *FontDialog* espone le proprietà:

- **MinSize** - ottiene o imposta la dimensione minima selezionabile da un utente, espressa in punti.
- **MaxSize** - ottiene o imposta la dimensione minima selezionabile da un utente, espressa in punti.
- **FontMustExist** - indica se nella finestra di dialogo viene descritta una condizione di errore quando l'utente cerca di selezionare un tipo di carattere o uno stile inesistente.
- **ShowEffects** - ottiene o imposta un valore che indica se nella finestra di dialogo sono inclusi controlli che consentono all'utente di specificare opzioni di testo quali il barrato, la sottolineatura e il colore.
- **AllowVerticalFonts** - ottiene o imposta un valore che indica se nella finestra di dialogo sono visualizzati sia tipi di carattere verticali sia orizzontali oppure soltanto orizzontali.
- **AllowVectorFonts** - ottiene o imposta un valore che indica se la finestra di dialogo consente di selezionare tipi di carattere vettoriali.

Ed infine la proprietà *Font*, che imposta o restituisce il font selezionato.

Il controllo PrintDialog

Il controllo *PrintDialog* permette di visualizzare la finestra di dialogo *Stampa* che consente all'utente di selezionare la stampante, il numero di copie e le pagine da stampare. Prima di utilizzare il controllo *PrintDialog*, è necessario inserire nella form il controllo *PrintDocument* ed associarlo al controllo *PrintDialog* tramite la proprietà *Document* (cliccando sulla freccia rivolta verso il basso accanto alla proprietà apparirà l'elenco degli oggetti *PrintDocument* selezionabili).
L'oggetto *PrintDocument* espone la proprietà *DocumentName* che "informa" la finestra di dialogo *Stampa* quale documento deve essere stampato.

Il controllo PageSetupDialog

Il controllo *PageSetupDialog* permette di visualizzare la finestra di dialogo *Imposta* pagina che consente all'utente di stabilire l'orientamento della pagina (verticale o orizzontale), le dimensioni del foglio e le dimensioni per i quattro margini della pagina. Anche in questo caso, prima di utilizzare il controllo *PageSetupDialog*, è necessario inserire nella form il controllo *PrintDocument* ed associarlo al controllo *PageSetupDialog* tramite la proprietà *Document*.

Ing. Luigi Buono

Visual Basic .NET

MSG Box

Un metodo alternativo per visualizzare le finestre di messaggio è l'utilizzo del comando *MsgBox*. Il comando *MsgBox* era il comando utilizzato in VB6 che è stato conservato anche in VB.NET. Le finestre messaggio prodotte utilizzando *MsgBox* appaiono identiche a quelle scritte con *MessageBox*, la differenza è tutta nella scrittura del codice. La sintassi è la seguente:

```
MsgBox(testo,
  setDibottoni, etichetta)
```

I parametri *testo* ed *etichetta* visualizzano il messaggio ed il titolo, così come in *MessageBox*, il parametro *setDiBottoni* ammette, invece, una combinazione dei parametri: *bottone*, *icona* e *bottoneDiDefault* separati dal segno più (+). Per visualizzare il messaggio di esempio riportato nell'articolo, utilizzando *MsgBox* si deve scrivere:

```
MsgBox("Vuoi salvare i
  dati prima di uscire ?",
  MsgBoxStyle.YesNoCancel
  + MsgBoxStyle.Question
      + MsgBoxStyle
      .DefaultButton3,
  "Richiesta Conferma")
```

C#

Passaggio

DI ARGOMENTI, TRE CASI PARTICOLARI

Il passaggio di argomenti ad un metodo, insieme con la ricezione dei valori di ritorno, costituisce uno dei capisaldi della programmazione orientata agli oggetti, tanto con C# quanto con altri linguaggi di programmazione. In questa lezione esamineremo tre interessanti tecniche messe a disposizione da C#, utili per il passaggio di argomenti in situazioni particolari.

File sul CD
\soft\codice
\codici_csharp.zip

Già abbiamo parlato di metodi e di passaggio di argomenti, in maniera elementare. In questa lezione andremo ad approfondire l'argomento, esaminando alcune tecniche utili per fornire argomenti in situazioni particolari.
Saranno esaminate le tre parole chiave *ref*, *out* e *params*.

PASSAGGIO DI ARGOMENTI CON REF

C#, come è stato già detto nel corso delle precedenti lezioni, distingue tra due macro-insiemi di dati: quelli gestiti per valore e quelli amministrati per riferimento. Il tipo di gestione attuato dal CLR di .NET costringe a delle riflessioni, soprattutto nel momento in cui un valore è trattato come argomento di ingresso di un qualsiasi metodo.
Riassumendo, i tipi valore sono sempre copiati e duplicati, mentre i tipi riferimento utilizzano alias differenti per indicare una medesima area della memoria.
Quando un tipo valore, ad esempio un *int*, è usato come argomento di ingresso, il CLR ne esegue automaticamente una copia da affidare al codice contenuto nel corpo del metodo che lo riceve.
Modificando la copia, l'originale rimane inalterato. E' possibile variare il comportamento predefinito che l'esecutore assume nei confronti dei tipi valore, servendosi della parola chiave *ref*. Prendiamo in esame il seguente esempio:

```
class ScambiaVariabili
{
  public static void scambiaSenzaRef(int a, int b)
  {
      int aux = a;
      a = b;
      b = aux;
  }

  public static void scambiaConRef(ref int a, ref int b)
  {
     int aux = a;
     a = b;
     b = aux;
  }

  public static void mostraValori(int i1, int i2)
  {
      System.Console.WriteLine("i1 vale " + i1);
      System.Console.WriteLine("i2 vale " + i2);
  }

  public static void Main()
  {
     int i1 = 5;
     int i2 = 3;
     mostraValori(i1, i2);

     scambiaSenzaRef(i1, i2);
     System.Console.WriteLine("Dopo
                                          scambiaSenzaRef");
     mostraValori(i1, i2);

     scambiaConRef(ref i1, ref i2);
```

```
    System.Console.WriteLine("Dopo scambiaConRef");
    mostraValori(i1, i2);
  }

}
```

La classe *ScambiaVariabili* comprende due metodi dal corpo identico. Il primo, *scambiaSenzaRef()*, è formulato in maniera classica:

```
public static void scambiaSenzaRef(int a, int b)
{
   int aux = a;
   a = b;
   b = aux;
}
```

All'interno del corpo del metodo avviene uno scambio tra i valori conservati in *a* ed in *b*. Tuttavia, questa operazione non ha alcun riscontro pratico. Gli interi *a* e *b*, infatti, sono sempre delle copie dei valori originariamente forniti al metodo.
Lo dimostra la prima parte dell'output restituito dal programma:

i1 vale 5
i2 vale 3
Dopo scambiaSenzaRef
i1 vale 5
i2 vale 3

Le variabili *i1* e *i2* dichiarate all'interno del metodo *Main()*, come è possibile osservare, non hanno subito modifiche. È più che lecito che sia così: *i1* e *i2* sono passate per valore, ogni modifica sulle copie non si riflette sugli originali. Diversamente avviene con il metodo *scambiaConRef()*, benché il corpo impiegato sia il medesimo:

```
public static void scambiaConRef(ref int a, ref int b)
{
  int aux = a;
  a = b;
  b = aux;
}
```

L'unica differenza riscontrabile è nella lista degli argomenti di ingresso. Alla dichiarazione dei due interi *a* e *b*, infatti, è stato fatto precedere lo specificatore *ref*. La stessa parola chiave deve essere nuovamente digitata nel momento in cui il metodo viene richiamato:

```
scambiaConRef(ref i1, ref i2);
```

In questo caso, gli interi *i1* e *i2* vengono trasferiti al metodo per riferimento. Il CLR non eseguirà delle copie. Pertanto, una modifica sul contenuto di *a* si rifletterà su *i1*, così come variando *b* si cambierà anche *i2*.
Lo dimostra l'output prodotto dal programma.
Dopo *scambiaConRef*

i1 vale *3*
i2 vale *5*

I valori conservati nelle variabili, in questo caso, sono stati invertiti. Il metodo, così formulato, ha finalmente un senso pratico, che altrimenti non avrebbe.

PASSAGGIO DI ARGOMENTI CON OUT

Gli argomenti forniti ad un metodo devono sempre essere inizializzati prima di essere forniti ad un metodo, indipendentemente dal loro tipo. In generale, vale sempre la regola *"prima dichiari, poi inizializzi e quindi utilizzi"*. Il passaggio di argomenti, in effetti, è un vero e proprio utilizzo delle variabili.
Senza l'inizializzazione, una variabile non può essere sfruttata, almeno in situazioni normali. Prendiamo in considerazione il seguente codice:

```
class OutTest1
{

 public static void somma(int a, int b, ref int c)
 {
    c = a + b;
  }

 public static void Main()
 {
    int a = 5;
    int b = 3;
    int c;
    somma(a, b, ref c);
    System.Console.WriteLine("c vale " + c);
 }

}
```

Non è possibile compilare questa classe, poiché l'intero *c* non è stato inizializzato prima di essere passato, per riferimento, al metodo *somma()*.
L'errore che se ne ricava è mostrato di seguito:

```
error CS0165: Utilizzo della variabile locale "c"
                                    non assegnata.
```

Nonostante questo, esistono dei casi in cui si può desiderare di fare uso di una variabile non

C#

Argomenti
Gli argomenti forniti ad un metodo devono sempre essere inizializzati prima di essere forniti ad un metodo, indipendentemente dal loro tipo.

C#

inizializzata. La seguente variante del precedente esempio ci introduce all'utilizzo della parola chiave *out*:

```
class OutTest2
{

 public static void somma(int a, int b, out int c)
 {
   c = a + b;
 }

 public static void Main()
 {
   int a = 5;
   int b = 3;
   int c;
   somma(a, b, out c);
   System.Console.WriteLine("c vale " + c);
 }

}
```

OutTest2 è stata ottenuta sostituendo la parola *out* alle clausole *ref* già presenti in *OutTest1*. Diventa ora possibile compilare ed eseguire il codice. Sostanzialmente, il passaggio di parametri attraverso la parola chiave *out* porta a due conseguenze:

1. La variabile fornita viene trasferita sempre e comunque per riferimento, anche se è un tipo valore.

2. Non è richiesto che la variabile fornita sia inizializzata, giacché *out* costituisce una tecnica che si affianca a *return* nella restituzione di valori al codice chiamante.

Ad ogni modo, perché si dovrebbe utilizzare *out* per la restituzione di valori, quando si ha già a disposizione il ben più comodo *return*? Semplicemente, potrebbe capitare che *return* non sia sufficiente. Con *return* è possibile restituire un solo dato alla volta, mentre con *out* è possibile avere quanti valori di ritorno si desiderano. Il vantaggio di *out*, dunque, non è evidente con il metodo *somma()*, giacché questo poteva più abilmente essere scritto nella forma:

```
public static int somma(int a, int b)
{
  return a + b;
}
```

Prendiamo ora in considerazione questa nuova classe:

```
class OutTest3
{
 public static void perimetroAndArea(int l, out int per,
                                              out int area)
 {
   per = l * 4;
   area = l * l;
 }
 public static void Main()
 {
   int l = 5;
   int p, a;
   perimetroAndArea(l, out p, out a);
   System.Console.WriteLine("Perimetro: " + p);
   System.Console.WriteLine("Area: " + a);
 }
}
```

Il metodo *perimetroAndArea()* calcola il perimetro e l'area di un quadrato di lato l, in un colpo solo. Impiegando *return*, non sarebbe stato possibile compiere entrambi i calcoli con una singola funzione: si sarebbe potuto restituire solamente uno dei due valori calcolati, l'altro sarebbe andato perso.
Ad ogni modo, *return* e *out* non si escludono a vicenda: in particolari situazioni, è possibile realizzare metodi che restituiscano un valore tramite *return* ed altri tramite *out*, simultaneamente e senza conflitti.

PASSAGGIO DI ARGOMENTI CON PARAMS

In tutti i casi esaminati sinora, il numero di argomenti forniti ad un metodo era sempre noto a priori. Ad esempio, il metodo *somma()* compreso in *OutTest2* accetta sempre tre argomenti: i due addendi e la variabile per la restituzione della loro somma. Ovviamente, *somma()* non può sommare più di due addendi alla volta. Può essere interessante, invece, gestire un numero di argomenti non noto a priori, ad esempio per sommare *n* addendi:

```
class ParamsTest
{

 public static int somma(params int[] addendi)
  {
   int totale = 0;
   for (int i = 0; i < addendi.Length; i++)
   {
      totale += addendi[i];
    }
   return totale;
 }

 public static void Main()
 {
```

Dispense Web

Tra le dispense Web del corso troverete appunti, aggiunte, approfondimenti, risposte a domande frequenti e altre appendici legate alla lezione odierna. L'indirizzo di riferimento è http://www.sauronsoftware.it/dispenseweb/csharp/

```
    int t1 = somma(5, 3);
    System.Console.WriteLine(t1);

    int t2 = somma(2, 5, 1, 10, 9);
    System.Console.WriteLine(t2);

    int t3 = somma(6, 3, 8);
    System.Console.WriteLine(t3);
  }

}
```

La clausola *params* permette di gestire un numero variabile di argomenti.
Le tre chiamate:

```
somma(5, 3);
somma(2, 5, 1, 10, 9);
somma(6, 3, 8);
```

sono tutte valide e riferite allo stesso metodo, benché differiscano nel numero degli argomenti specificati.
Con la dicitura:

```
params int[] addendi
```

abbiamo fatto in modo che il metodo *somma()* dell'esempio corrente possa gestire un numero imprecisato di argomenti di tipo *int*. Di fatto, quello che viene fornito al corpo del metodo è un array di valori *int*. Questo corso ancora non ha esaminato nel dettaglio gli array. Per il momento, è sufficiente sapere che un array è una collezione di valori del medesimo tipo.
Con la proprietà:

```
nomeArray.Length
```

è possibile sapere quanti elementi ci sono nell'insieme. Ad ogni elemento di un array corrisponde un indice numerico, da zero in poi. Il primo elemento di un array ha indice 0, il secondo 1, il terzo 2 e così via. L'ultimo elemento ha sempre indice *Length - 1*. I singoli elementi della collezione possono essere selezionati attraverso una coppia di parentesi quadre:

```
nomeArray[0]
nomeArray[1]
nomeArray[2]
...
nomeArray[nomeArray.Length - 1]
```

In seguito, analizzeremo più nel dettaglio le collezioni e gli array, giacché questi costituiscono una categoria di oggetti molto importante per la programmazione di applicazioni raffinate.

CONCLUSIONI

C# è un linguaggio molto ricco, benché semplice. Esistono numerosi stratagemmi per abbreviare le operazioni di stesura del codice. Le tre caratteristiche esaminate in questa lezione, ad esempio, non sono indispensabili in un linguaggio orientato agli oggetti di derivazione Java, giacché possono essere rimpiazzate da tecniche alternative più complesse, ma lo stesso efficaci. Proprio il linguaggio Java, ad esempio, non comprende nessuna delle tre parole chiave esaminate in questa sede, poiché i progettisti di Sun le hanno giudicate ridondanti (e confusionarie).
Più in generale, quindi, C# deve molto all'esperienza di Java, ma trae anche spunto dalle abitudini di C e C++, che gli sviluppatori hanno imparato ad apprezzare nel corso degli anni.

Carlo Pelliccia

C# Guida per lo sviluppatore

ISBN: 88-203-2962-X
Autore: *Simon Robinson* e altri
Editore: Hoepli
Pagine: 1188
Costo: € 55,78
Target: programmatori abbastanza esperti

Con l'avvento di .NET, gli scaffali delle librerie cominciano a popolarsi delle prime edizioni dei manuali dedicati alla nuova piattaforma Microsoft. C#, ovviamente, è uno degli argomenti più discussi e richiesti dalla comunità degli sviluppatori.
È comprensibile: conoscere C#, di fatto, significa avere la padronanza di tutti i meccanismi del nuovo framework, giacché al suo interno sono contenute le strutture che meglio astraggono le funzionalità offerte da .NET. E poi, non dimentichiamolo, C# è un linguaggio nuovo (almeno nel nome ed in alcune caratteristiche), che incuriosisce e desta l'interesse dell'intera comunità degli sviluppatori Windows.
Il manuale in questione si rivolge ai programmatori mediamente esperti che già vantino nel loro curriculum esperienze con linguaggi analoghi a C#, soprattutto Java e C++. Il volume, come è lecito attendersi, presenta nei primi capitoli un'introduzione ai concetti di base del nuovo framework. Si parla quindi di .NET, di linguaggio interno, di codice gestito, di assembly e di altro ancora. Viene poi introdotto il linguaggio C#. In tutti i capitoli più espressamente dedicati alle strutture del linguaggio, gli autori si sforzano di illustrare la programmazione orientata agli oggetti anche a chi proviene da un background differente, come i programmatori Visual Basic.

C#

Bibliografia

• GUIDA A C#
Herbert Schildt
(McGraw-Hill)
ISBN 88-386-4264-8
2002

• INTRODUZIONE A C#
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• C# GUIDA PER LO SVILUPPATORE
Simon Robinson e altri
(Hoepli)
ISBN 88-203-2962-X
2001

C++

Risoluzione

DI AMBIGUITÀ ED EREDITARIETÀ MULTIPLA

Abbiamo visto una panoramica del concetto di ereditarietà, che rappresenta una delle differenze sostanziali tra programmazione procedurale e programmazione Object-Oriented, nonché una delle caratteristiche più importanti degli oggetti. Adesso scenderemo un po' più nel dettaglio con nuovi concetti.

File sul CD
\soft\codice\C++.zip

Nel precedente appuntamento si è esaminato come si possa ridefinire nella classe derivata ognuna delle funzioni particolari, tipiche di una classe (cioè il costruttore, il distruttore e l'operatore di assegnazione), facendo eventualmente uso di quelle della classe base. Adesso, siccome siamo curiosi, ci chiediamo: possiamo ridefinire, nella classe derivata, una qualunque delle funzioni della classe base? Sebbene apparentemente sembri un problema banale, in cui la risposta affermativa ci sorge spontanea (quantomeno per trasporto emotivo), tuttavia l'apparenza, al solito, inganna: la soluzione praticamente banale è però molto insidiosa, e solo il bravo programmatore C++ sa come evitarne i pericoli. Supponiamo di avere le (solite) classi *Scheda* e *SchedaAvanzata*, e supponiamo che la derivazione tra le due sia di tipo *public*, mentre i campi *nome* e *telefono* siano stati dichiarati *protected* in *Scheda*: queste ultime ipotesi ci servono solo per semplificare il discorso, non sono necessarie. Il risultato delle nostre precondizioni, che supporremo verificate d'ora in avanti, è che i campi *nome* e *telefono* della classe *Scheda* sono a questo punto ereditati come protetti nella classe *SchedaAvanzata* (se necessario, ricontrollate lo schema introdotto la scorsa puntata).
A questo punto, ignorando la possibilità di ridefinire l'operatore <<, decidiamo di aggiungere un metodo alla classe *Scheda* che si occupi di stampare, con opportuna formattazione, il suo oggetto di invocazione. Una possibile definizione potrebbe essere:

```
//nel file scheda.cpp
void Scheda::StampaFormattata()
{           cout << "+-----------------------+\n";
            cout << "|      " << nome << "     |\n";
            cout << "+-----------------------+\n";
            cout << " telefono: " << telefono << "\n";
```

In questa funzione, abbiamo trascurato la lunghezza della stringa *nome*, supponendo che sia lunga il giusto per apparire centrata nell'etichetta: in realtà dovremmo gestire da noi la cosa, ricavandoci la lunghezza di tale stringa e completando la stampa, con l'inserimento di opportuni caratteri di spaziatura oppure allungando l'etichetta del necessario. Potrebbe essere un utile esercizio per il lettore volenteroso. Il metodo appena scritto (supposto *public* nella classe *Scheda*) viene adesso ereditato dalla classe *SchedaAvanzata*, rendendo così possibili operazioni del tipo:

```
Scheda pierino("Pierino","61010610rimarrai");
SchedaAvanzata piero("Piero Pierrot", "0123456789",
                                   "pierrot@maschere.it");
pierino.StampaFormattata();
piero.StampaFormattata();
```

Si presenta tuttavia adesso un problema. Quando abbiamo deciso di creare la classe *SchedaAvanzata*, non avevamo l'intenzione di creare una nuova classe svincolata dalla precedente: la nostra intenzione era di estendere la classe *Scheda*, usandola come base per la nuova classe. L'intenzione era poter inserire in un oggetto *SchedaAvanzata* ulteriori informazioni oltre quelle inseribili in un analogo oggetto *Scheda*. Se adesso usiamo per la stampa formattata il metodo appena definito, il contenuto informativo che ha in più un oggetto *SchedaAvanzata* rispetto ad un analogo oggetto *Scheda* è come se non esistesse: in altre parole *Scheda* e *SchedaAvanzata* dal punto di vista di questo metodo risultano indistinguibili, e il contenuto di un oggetto *SchedaAvanzata* non è visualizzato in maniera completa (in quanto manca la stampa del valore del campo *email*). Non possiamo usare il campo *email* nel nostro metodo, perché esso fa parte della classe derivata (*SchedaAvan-*

zata) e quindi nella classe base (*Scheda*) non esiste; non possiamo semplicemente spostare il metodo, opportunamente modificato per la stampa del campo mancante, nella classe derivata, poiché si perderebbe la funzionalità, rappresentata dal metodo, nella classe base (in altre parole, potremmo stampare con questo metodo solo oggetti di tipo *SchedaAvanzata*, ma non esisterebbe un analogo metodo per gli oggetti *Scheda*). Potremmo scrivere un metodo apposito per la classe derivata, ma (sebbene corretto) si costringerebbe l'utente della classe derivata e della classe base a ricordare la particolarità: è uno sforzo mnemonico superfluo. Possiamo però fare la cosa giusta, oltre che auspicabile ed intuitiva: ridefiniamo nella classe derivata lo stesso metodo della classe base.

UN ESEMPIO DI RIDEFINIZIONE

La ridefinizione si usa ogni volta che si vuole far corrispondere una funzione della classe derivata ad una funzione della classe base, della quale l'omonima appartenente alla classe derivata costituisce una specializzazione. Quando ridefiniamo una funzione nella classe derivata, essenzialmente basta che ci comportiamo come se non esistesse un analogo metodo nella classe base. Scriviamo così il nostro "nuovo" metodo nella classe derivata:

```
//nel file scheda_adv.cpp
void SchedaAvanzata::StampaFormattata()
{           cout << "+------------------------+\n";
            cout << "|      " << nome << "      |\n";
            cout << "+------------------------+\n";
            cout << " telefono: " << telefono << "\n";
            cout << "    email: " << email << "\n"; }
```

A questo punto, il codice di prova scritto nel paragrafo precedente funziona in maniera corretta, stampando il contenuto del campo *email* per (e solo per) gli oggetti di tipo *SchedaAvanzata*. Si deve, però, prestare attenzione ad un aspetto relativamente a questo meccanismo di ridefinizione: quando ridefiniamo un metodo nella classe derivata, essa maschera nella classe derivata tutte le funzioni ad essa omonime presenti nella classe base (cioè, il mascheramento avviene solo guardando il nome della funzione, senza guardare le liste dei parametri: in altre parole, non è un overload!). Ad esempio, avendo una generica classe *X* con la seguente dichiarazione:

```
class X
{ public:
        F(int);
        F(char); };
```

e definendo una classe derivata *Y* nel seguente modo:

```
class Y: public X
{ public:
        F(); };
```

si avrebbe che il metodo *Y::F()* maschera entrambi i metodi *X::F(int)* ed *X::F(char)*, sebbene abbiano tra loro liste di parametri diverse. Quindi il seguente codice darebbe un errore:

```
Y dummy;
dummy.F(10); //sbagliato: il metodo invocato non esiste!
```

e questo perché il metodo *F(int)*, sebbene presente nella classe base, è diventato invisibile nella classe derivata in quanto mascherato dal metodo *Y::F()*, senza parametri. I metodi della classe base non sono, però, perduti per sempre: per accedervi basta usare l'operatore di *scope resolution*, *"::"*. Tale operatore, incontrato già parlando delle definizioni delle funzioni di una classe, chiarisce al compilatore quale sia la classe di appartenenza di un metodo chiamato: l'uso di questo operatore non è solo relegato alle definizioni delle funzioni di una classe! Infatti, questo operatore può essere usato ogni volta si renda necessario specificare una cosa del tipo "voglio che venga invocato il metodo *M* della classe C". Questo è proprio ciò che ci serve per accedere (da un oggetto della classe derivata) ai metodi della classe base una volta che essi siano stati ridefiniti nella classe derivata. Possiamo infatti accedere al metodo *X::F(int)* anche da un oggetto della classe *Y*, semplicemente scrivendo:

```
Y dummy;
dummy.X::F(10);    //giusto!
```

In questo modo si è inteso dire che l'oggetto di invocazione dummy richiama il metodo *F(int)* della classe (base) *X*. Ovviamente, il metodo invocato tratterà adesso l'oggetto *dummy* (appartenente alla classe derivata) come fosse un oggetto della classe (base) *X*: questo significa che eventuali campi specifici della classe derivata non saranno considerati. Tornando al caso specifico delle classi *Scheda* e *SchedaAvanzata*, potremmo quindi scrivere:

```
SchedaAvanzata XMen("Wolverine","1235813",
                                        "io@wolverine.tv");
XMen.Scheda::StampaFormattata();
```

ottenendo quindi una stampa a schermo come se *X-Man* fosse un oggetto della classe (base) *Scheda* invece che della classe (derivata) *SchedaAvanzata*: il campo email viene, semplicemente, ignorato dalla funzione invocata.

EREDITARIETÀ MULTIPLA

Una caratteristica molto interessante della programmazione a oggetti, che il C++ implementa, è l'ereditarietà multipla. Con questo nome si fa riferimento al

Contatta gli autori!

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@libero.it (Alfredo)

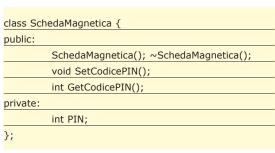
e

marcodelgobbo@libero.it (Marco)

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Fig. 1: **Catena di derivazione.**

meccanismo per il quale è possibile derivare una classe da più di una classe base, direttamente. Per "direttamente" si intende il fatto che la derivazione avviene nell'arco di un solo livello di ereditarietà, e non nell'ambito di una catena di derivazioni. Questo significa che le diverse classi base di una unica classe derivata possono anche essere completamente slegate tra loro e non avere niente in comune (anzi questo è proprio il caso più comune e utile in pratica). Per chiarire meglio questo concetto supponiamo di avere una classe *SchedaAvanzatissima* derivata da *SchedaAvanzata*. La catena di derivazioni è mostrata in Fig. 1.a. In questo caso NON si tratta di ereditarietà multipla, in quanto, come si può notare, tutte le classi che sono derivate di altre (cioè *SchedaAvanzata* e *SchedaAvanzatissima*) hanno una sola classe base. Il caso dell'ereditarietà multipla è esemplificato invece in Fig. 1.b. In questo caso abbiamo due classi base (*SchedaAvanzata* e *SchedaMagnetica*) dalla quale deriva la classe *SchedaMagneticaAvanzata*. Ma come si comporta un oggetto di una classe derivata tramite ereditarietà multipla? Molto probabilmente il suo comportamento è intuibile dai concetti già appresi sull'ereditarietà. Infatti, così come una classe derivata in maniera standard racchiude ed estende le caratteristiche della sua classe base, allo stesso modo una classe ottenuta mediante ereditarietà multipla, avrà i campi e le funzioni di tutte le sue classi base, esattamente come se ereditasse singolarmente da ciascuna di esse. Per fare un esempio, supponiamo che *SchedaMagnetica* sia definita nel seguente modo:

```
class SchedaMagnetica {
public:
        SchedaMagnetica(); ~SchedaMagnetica();
        void SetCodicePIN();
        int GetCodicePIN();
private:
        int PIN;
};
```

Definendo semplicemente *SchedaMagneticaAvanzata* come derivata di *SchedaMagnetica* e *SchedaAvanzata*, senza definire altri nuovi campi o funzioni, cioè nel seguente modo:

```
class SchedaMagneticaAvanzata : public
SchedaMagnetica,SchedaAvanzata{ //... qui nulla... };
```

potremmo istanziare un oggetto che ci renda disponibili i campi delle due classi:

```
SchedaMagneticaAvanzata sma;
sma.ImpostaNome("Attarico Il Babbaro");
                                  //funzione di SchedaAvanzata
sma.SetCodicePIN("0123"); //funzione di SchedaMagnetica
```

in questo modo avremmo una classe che ingloba le altre due in maniera grezza, senza cioè offrire caratteristiche aggiuntive o funzionalità ulteriori rispetto a quelle che avremmo istanziando due oggetti diversi. Tuttavia, a volte, questo è utile ed è proprio quello che si vuole. Analogamente a quanto accade per l'ereditarietà semplice, anche per quella multipla è possibile utilizzare la compatibilità degli oggetti, cioè a dire che un oggetto derivato può essere utilizzato in ogni punto in cui è richiesto un oggetto della classe base:

```
void FunzioneDummy1(SchedaAvanzata s) {...}
void FunzioneDummy2(SchedaMagnetica s) {...}
SchedaMagneticaAvanzata sma;
FunzioneDummy1(sma); //uso sma come SchedaAvanzata
FunzioneDummy2(sma); //uso sma come SchedaMagnetica
```

Ovviamente, come abbiamo già visto, non è possibile invertire il verso di questa compatibilità.

ULTERIORI AMBIGUITÀ

Cosa succede nel caso in cui le classi base hanno due funzioni con lo stesso nome? Quale delle due verrà "ereditata" dalla classe derivata e utilizzata quando viene invocata da un suo oggetto? Ci troviamo di fronte a un caso di ambiguità simile a quello trattato in precedenza, che deve essere risolto in un qualche modo. Supponiamo, per fare un esempio, che sia *SchedaAvanzata* che *SchedaMagnetica* abbiano la seguente funzione membro:

```
void ImpostaValidita(bool valida);
```

Scrivendo questo codice:

```
SchedaMagneticaAvanzata sma;
sma.ImpostaValidita(true);
```

quale funzione verrà chiamata? Quella di *SchedaAvanzata* o quella di *SchedaMagnetica?* In realtà in questo caso non esiste alcuna discriminante valida per effettuare una decisione, pertanto il compilatore segnalerà un errore in questo punto. Si badi bene che anche questo è un caso di mascheramento, per cui il compilatore segnalerebbe errore anche qualora le funzioni *ImpostaValidita()* delle classi base avessero una lista di parametri differente: nel mascheramento conta solo il nome del campo. Cosa si può fare per ovviare a questa situazio-

Avviso di chiamata

Nell'articolo sia le funzioni *SchedaAvanzata::ImpostaValidità()* che *SchedaMagneticaAvanzata::ImpostaValidità()* hanno lo stesso comportamento, quindi la differenza tra il chiamare l'una o l'altra potrebbe non essere molto evidente. Per questo abbiamo inserito nel codice che trovate nel CD allegato, delle stampe a schermo che chiariranno l'effettivo percorso delle chiamate effettuate. Consigliamo di dare un'occhiata!

ne? Esistono diverse strategie. Una prima possibilità è quella di utilizzare, come visto in precedenza nel caso dell'ereditarietà semplice, l'operatore di scope resolution "::" e scrivere, ad esempio:

```
SchedaMagneticaAvanzata sma;
sma.SchedaAvanzata::ImpostaValidita(true);
```

così facendo il compilatore avrà le idee chiare su cosa vogliamo intendere. Una seconda possibilità, forse più adatta a questo contesto, è quella di ridefinire nella classe *SchedaMagneticaAvanzata* la funzione *ImpostaValidita()* in modo che faccia esattamente ciò che desideriamo; ad esempio potrebbe chiamare il relativo metodo di una delle sue classi base:

```
void SchedaMagneticaAvanzata::ImpostaValidita(bool v)
{ SchedaAvanzata::ImpostaValidita(v); }
```

o ancora potrebbe ridefinire completamente il comportamento della funzione in questione, magari facendole impostare un nuovo campo privato di tipo booleano:

```
void SchedaMagneticaAvanzata::ImpostaValidita(bool v) {
   valida = v; //valida è dichiarato private in
                                     SchedaMagneticaAvanzata }
```

Quest'ultima soluzione è senz'altro da preferire dal punto di vista semantico, in quanto il concetto di "validità" di una scheda va al di là del fatto che si tratti di una scheda magnetica o quant'altro: una scheda è valida oppure non lo è, indipendentemente dalla sua natura.

FUNZIONI VIRTUALI

Abbiamo visto in precedenza come, per il concetto di compatibilità tra oggetto base e oggetto derivato, sia possibile usare quest'ultimo laddove è richiesto il primo. Detta così sembra la cosa più banale del mondo, ma cosa succede nel seguente caso?

```
void RendiValida(SchedaAvanzata& s) {
          s.ImpostaValidita(true);
}
SchedaMagneticaAvanzata sma;
RendiValida(sma);
```

Si utilizza in questo caso un oggetto di tipo *SchedaMagneticaAvanzata* al posto di uno di tipo *SchedaAvanzata*. All'interno di *RendiValida()* è chiamata *ImpostaValidità()* e, si noti bene, il parametro è passato per riferimento, quindi non entra in gioco alcun costruttore di copia. Verrà quindi chiamata la funzione *ImpostaValidità()* di *SchedaAvanzata* (che è il tipo di oggetto che ci aspettiamo venga passato quando scriviamo la funzione *RendiValida()*) o quella di *SchedaMagneticaAvanzata()* (che maschera l'altra)? La risposta è che verrà invocata la funzione di *SchedaAvanzata*, in quanto, per le funzioni normali, il binding (cioè la scelta di quale funzione chiamare di volta in volta) viene fatto al momento di compilare la funzione (si parla di *early binding*), quando cioè non si sa di quale tipo sarà l'oggetto passato, o meglio, si presume ragionevolmente che esso sia del tipo specificato nei parametri; in questo caso, appunto, *SchedaAvanzata*. Ma come fare allora a utilizzare una funzione ridefinita in una classe derivata (che presumibilmente è stata ridefinita proprio perché quella base non era adatta o sufficiente), sfruttando al tempo stesso la compatibilità? Bisogna introdurre un altro concetto: quello di funzione virtuale. Dichiarare una funzione come virtuale, significa dire al compilatore: "Quando incontri una chiamata a questa funzione non compilare subito il codice relativo, ma aspetta che venga deciso a tempo di esecuzione qual'è la funzione giusta da agganciare qui". Il decidere a tempo di esecuzione la funzione da utilizzare si chiama, in letteratura informatica, late binding. Sfruttando il late binding in *RendiValida()* il codice:

```
SchedaMagneticaAvanzata sma;
RendiValida(sma);
```

chiama effettivamente la funzione *ImpostaValidita()* di *SchedaMagneticaAvanzata*, poiché di questo tipo è il parametro passato.

Per utilizzare il late binding è necessario che *ImpostaValidita()* in *SchedaAvanzata* sia dichiarata virtuale, tramite la parola chiave *virtual*:

```
//nel file scheda_adv.h
virtual void ImpostaValidita(bool valida);
```

È quindi necessario in ogni caso fare uno sforzo di lungimiranza e prevedere a priori quali potranno essere le funzioni soggette a ridefinizione, per le quali è necessaria la dichiarazione virtual. Anche in questo caso comunque è bene non cadere nella sindrome del "dichiaro tutto virtual" in quanto, il late binding è in ogni caso un meccanismo attuato a tempo di esecuzione, per cui meno ottimizzato e che potrebbe portare a delle inefficienze.

CONCLUSIONI

In questa lezione abbiamo trattato alcuni degli argomenti più interessanti dell'ereditarietà in C++. Tuttavia non è ancora tutto in quanto questo meccanismo consente delle vere e proprie raffinatezze stilistiche e concettuali degne dei migliori programmatori. Noi ve le sveleremo nel prossimo appuntamento in cui parleremo, tra l'altro di classi base astratte e funzioni virtuali pure. Alla prossima!

Marco Del Gobbo & Alfredo Marroccelli

Funzioni virtuali

Dichiarare una funzione come virtuale, significa dire al compilatore: "Quando incontri una chiamata a questa funzione non compilare subito il codice relativo, ma aspetta che venga deciso a tempo di esecuzione qual'è la funzione giusta da agganciare qui".

Java

Java

MEDIA FRAMEWORK: STRUMENTI DI ACQUISIZIONE AUDIO E VIDEO

Non bisogna essere degli esperti programmatori per realizzare applicazioni che facciano uso di dispositivi di acquisizione audio e video. Il pacchetto Java Media Framework mette a disposizione tutto quello che è necessario per riuscirci scrivendo soltanto qualche linea di codice!

File Sul CD
\soft\codice \JMFSourceCode2.zip

La semplicità di utilizzo degli strumenti presenti nei packages JMF è tale che per realizzare un multimedia player non è necessario essere esperti programmatori! Bastano poche istruzioni ed il gioco è fatto:

- selezionare il file da riprodurre;
- ottenere l'url corrispondente al percorso;
- costruire un player controller per il file;
- aggiungere alla finestra il componente di controllo e di riproduzione del player;
- attivare, infine, il player!

Per implementare i precedenti passi è possibile utilizzare come riferimento il seguente blocco di codice (tratto dal metodo *buildUI()* della classe *Esempio1* allegata).

```
...
Player player;
JFileChooser fileChooser = new JFileChooser(".");
int status = fileChooser.showOpenDialog(this);
if (status == JFileChooser.APPROVE_OPTION)
{ File file = fileChooser.getSelectedFile();
   try{URL url = file.toURL();
         f inal Container contentPane = getContentPane();
         player = Manager.createRealizedPlayer(url);
         //A
         Component visualComponentUI =
                                    player.getVisualComponent();
         if (visualComponentUI != null)
          contentPane.add( visualComponentUI,
                                          BorderLayout.CENTER);
         Component controlPanelComponent =
                                 player.getControlPanelComponent();
         if (controlPanelComponent != null)
         contentPane.add( controlPanelComponent,
                                          BorderLayout.SOUTH);
         player.start();
         //A
   }
   catch(Exception e){System.out.println(e);
                                          System.exit(-1);}
}//if
...
```

La classe *Manager* mette a disposizione due diversi metodi statici per costruire un Player e cioè *Manager.createPlayer(…)* e *Manager.createRealizedPlayer (…)*. La differenza sostanziale sta nella modalità di controllo sulla costruzione del Player. Quando usiamo il metodo *Manager.createPlayer(…)* quello che otteniamo è soltanto il riferimento ad un oggetto *Player* che in realtà può ancora essere in costruzione, in quanto il metodo non è bloccante: nel caso in cui il *Player* non fosse ancora pronto, la successiva richiesta del componente di riproduzione video produrrà un errore *javax.media.NotRealizedError.* In realtà questa situazione non deve considerarsi un limite. La completa realizzazione dell'oggetto è segnalata da un evento di tipo *RealizeCompleteEvent:* basterà quindi realizzare un *ControllerAdapter* per questo evento e inserire all'interno del metodo *realizeComplete(RealizeCompleteEvent event)* la sezione delimitata dal commento *//A* (vedere esempio proposto nel precedente articolo). L'altro metodo, *Manager.createRealizedPlayer(…)* è invece bloccante: soltanto dopo la costruzione dell'oggetto verrà restituito il controllo al thread di esecuzione dell'applicazione.

Processor

Un *Processor* è un particolare tipo di *Player* che consente di ottenere come output i dati pre-elaborati per la riproduzione, naturalmente oltre a controllare e gestire la riproduzione dei contenuti multimediali. Una volta istanziato il *Processor,* mediante il metodo *getDataOutput()* è possibile avere il riferimento al *DataSource* dei dati che vengono prodotti in uscita.

CAPTUREDEVICE

Limitarci a riprodurre file multimediali è però una forte limitazione nell'utilizzo dei packages JFM. La realtà

che ci circonda è una fonte inesauribile di informazioni che spesso vogliamo condividere con gli altri. Possiamo utilizzare quello che ci sta intorno come "sorgente" dei contenuti audio e video delle applicazioni, acquisendoli mediante particolari dispositivi come telecamere digitali, webcam e microfoni. I dati ottenuti possono essere trattati, o direttamente memorizzati su disco in formato media (AVI, QUICKTIME, MP3, etc.) ovvero trasmessi attraverso la rete (*streaming*). Ciascun dispositivo hardware viene indicato come *CaptureDevice* e suddiviso in due categorie di sorgenti: di tipo *push* e di tipo *pull*. La differenza è molto semplice: si parla di *push source* quando la sorgente è di tipo continua (ed anche sullo stream si avrà un flusso continuo di dati), come ad esempio lo sono i microfoni. Una macchina fotografica digitale è invece una sorgente *pull* in quanto produce dati "isolati", in singole immagini (scatti). Utilizzando Java Media Framework, tutti i capture device assumono un livello di astrazione tale da trascurare completamente la loro natura hardware, consentendoci di individuarli soltanto mediante i formati dei dati prodotti. Ciascun dispositivo viene "mappato" su un generico *CaptureDevice* di cui ne conosciamo solamente le caratteristiche fondamentali a cui possiamo accedere usando il corrispondente *CaptureDeviceInfo*. Come nel caso visto in precedenza, cioé della riproduzione di file multimediali, anche in questo caso le istruzioni da seguire sono veramente semplici:

- localizzare il *CaptureDevice* mediante apposita richiesta al *CaptureDeviceManager*;
- dal device risalire alle sue caratteristiche mediate il suo *CaptureDeviceInfo*;
- sfruttando questo oggetto otterremo il *MediaLocator* necessario per creare il *DataSource* da cui attingere i dati prodotti;
- costruire il Player su questo *DataSource*;
- infine, avviare il processo di acquisizione.

Mediante chiamata al metodo statico *getDeviceList(…)* al*CaptureDeviceManager* è possibile ottenere la lista dei dispositivi disponibili (o meglio dei corrispondenti *CaptureDeviceInfo*),

```
…
java.util.Vector deviceList;
CaptureDeviceInfo captureDeviceInfo;
deviceList = CaptureDeviceManager.getDeviceList(null);
if (deviceList.size()>0)
    captureDeviceInfo = (CaptureDeviceInfo)
                              deviceList.firstElement();
…
```

oppure accedere direttamente ad un particolare dispositivo direttamente dal suo identificativo, sfruttando il metodo *getDevice(deviceName)*. Il metodo *getDeviceList(…)* ha come parametro di richiesta un *Format*, cioè un oggetto con il quale viene identificato, appunto, un particolare formato e che contiene tutte le informazioni necessarie al trattamento dei dati. Lasciando indeterminato questo parametro, il *CaptureDeviceManager* restituirà la lista completa di tutti i dispositivi disponibili, altrimenti fornisce soltanto l'elenco dei *CaptureDeviceInfo* corrispondenti ai dispositivi che possono produrre dati nel formato richiesto. La classe *Format* ha due classi derivate principali: una per definire il generico formato audio *(AudioFormat)*, l'altra per quello video *(VideoFormat)*. Ciascuna di queste classi viene poi utilizzata per specificare ogni formato (rispettivamente audio e video) supportato dalle librerie JFM. Ad esempio, possiamo richiedere al *CaptureDeviceManager* soltanto i device di acquisizione video: per farlo specifichiamo un formato audio generico, senza cioé specificare un particolare encoding type:

```
…
AudioFormat audioFormat = new AudioFormat(null);
…
```

ed utilizziamo l'oggetto *Format* come parametro al metodo *getDeviceList(…)*

```
…
deviceList=CaptureDeviceManager.getDeviceList
                                   (audioFormat);
for (int index=0; index<listDevice.size(); index++)
  {deviceInfo = (CaptureDeviceInfo)
                      listDevice.elementAt(index);
   System.out.println(deviceInfo.getName());
  }
…
```

Un'operazione analoga possiamo realizzarla per conoscere i dispositivi in grado di produrre dati in formato video: basterà sostituire l'oggetto *AudioFormat* con un *VideoFormat*. Una volta individuato il device che interessa, il passo successivo sarà quello di ottenere il suo *MediaLocator* da cui costruire il *DataSource* di produzione dei dati necessario al *Player*

```
…
MediaLocator mediaLocator = deviceInfo.getLocator();
DataSource dataSource = Manager.createDataSource(
                                   mediaLocator);.
Player player = Manager.createPlayer(dataSource);
…
```

oppure, in maniera più compatta

```
…
Player player = Manager.createPlayer(
                        deviceInfo.getLocator());
…
```

ma questa forma non è sempre utilizzabile. Cosa succede quando abbiamo contemporaneamente una sorgente audio ed una video (magari proveniente da due

Java

DataSource

Prendiamo una sorgente di dati "digitali": il flusso prodotto viene incapsulato mediante un oggetto *DataSource*, che rappresenta quindi i vari media audio, video o combinazione di entrambi. Una sorgente può essere un file o uno stream via Internet: specificando la locazione (mediante un semplice oggetto URL) oppure il protocollo (ad esempio *file://*) è possibile costruire il corrispondente *DataSource* da passare in ingresso al Player. Quando la sorgente dei dati proviene da uno stream via Internet, è bene effettuare una distinzione tra due tipi di *DataSource*:

1) Pull data source: lo si ha quando è il client che inizializza il trasferimento dei dati e ne controlla il flusso operando direttamente sulla sorgente. Ad esempio, i protocolli *HTTP* e *FILE* sono tipici casi di questo tipo di *DataSource*.

2) Push data source: in questo caso è il server che gestisce il flusso ed il client può soltanto acquisire i dati senza aver un controllo diretto su di essi, come nei sistemi di broadcasting e video on demand.

Java

DataSink

Il *DataSink* è l'interfaccia di base utilizzata per realizzare un particolare tipo di oggetti, in grado di recuperare i dati presenti su di un flusso e re-indirizzarli verso una qualche destinazione. Un *DataSink* può ad esempio recuperare i dati da un *DataSource* in uscita da un *Processor* e memorizzarli in un file multimediale (come visto nell'esempio proposto)

Formati supportati da JMF 2.1.1

La versione 2.1.1 del package supporta diversi tipi di media tra cui:

- **protocolli: FILE, HTTP, FTP, RTP**
- **audio: AIFF, AU, AVI, GSM, MIDI, MP2, MP3, QT, RMF, WAV**
- **video: AVI, MPEG-1, QT, H.261, H.263**

JMF fornisce anche il supporto pure a formati "concorrenti" come ad esempio Flash 2 e HotMedia.

Player

Un Player è un particolare oggetto che rende "riproducibile" i dati provenienti da stream di dati audio/video in ingresso. Questo processo di trasformazione è necessario per problemi di compatibilità con i dispositivi di riproduzione. Prendiamo ad esempio un CD audio. Per porer essere riprodotto, il player legge in input le sequenze di bit dal supporto ottico e le trasforma in un formato media compatibile con la scheda audio. Poiché i dispositivi sia di riproduzione che di acquisizione vengono trattati in maniera generica, il *Manager* deve riconoscere il formato dei dati da riprodurre (distinguendo tra i diversi formati audio e video) e costruire un *Player* compatibile. Il comportamento di questo *Controller* è scandito in base al particolare stato in cui si trova:

1) **UNREALIZED. Questo stato sta ad indicare che il *Player* è in fase di istanziazione, in quanto il *Manager* deve prima identificare il formato dei dati da trattare. Come già detto in precedenza, se il *Player* si trova in questo stato non può ancora essere utilizzato. Bisogna aspettare la conclusione delle operazioni di preparazione.**
2) **REALIZING. Dopo che il Manager ha determinato il formato dei dati su cui si sta costruendo il *Player*, lo stato dell'oggetto viene commutato mediante una chiamata al metodo *realize()*, che ne attiva la procedura di acquisizione delle risorse necessarie.**
3) **REALIZED. Completata la fase di preparazione, il Player passa in questo stato, comunicandolo mediante un evento *RealizeCompleteEvent*. A questo punto il Player è pronto, correttamente istanziato sul formato dei dati e con i necessari componenti visuali e *Controls*.**
4) **PREFETCHING. Chiamando il metodo *prefetch()* il *Player* comincia a prepararsi per la riproduzione, precaricando i dati, ottenendo l'uso esclusivo delle risorse necessarie.**
5) **PREFETCHED. Completato anche il precaricamento e l'allocazione delle risorse, il *Player* è finalmente pronto.**
6) **STARTED. Dallo stato *Prefetched* passa in *Started* appena viene chiamato il metodo *start()*.**

diversi dispositivi)? In teoria si potrebbe costruire un *Player* per ciascun *MediaLocator* (e quindi con il corrispondente *DataSource*) in modo da controllarne separatamente la riproduzione, ed in questo caso possiamo far uso della forma compatta:

```
…
    Player videoPlayer = Manager.createPlayer
                              (videoDevice.getLocator());
    Player audioPlayer = Manager.createPlayer
                              (audioDevice.getLocator());
    videoPlayer.addControllerListener(this);
    videoPlayer.start();
    audioPlayer.start();
```

Per avviare la riproduzione è necessario operare separatamente su i due Player invocando il metodo *start()*. In alternativa è possibile metterne uno sotto il controllo dell'altro

```
…
 videoPlayer.addController(audioPlayer);
 videoPlayer.start();
…
```

e quindi attivare la riproduzione sincronizzata di entrambi operando soltanto su di uno. Ma se invece di avere due distinti *DataSource* si volesse operare su di un unico stream, è possibile chiamare il metodo *createMergingDataSource(DataSource[] dataSources)* sul *Manager* per ottenere una combinazione (nel nostro caso tra audio e video). Naturalmente la forma compatta non è più applicabile, in quanto il *Player* va costruito sul *DataSource* combinato e non più sul *MediaLocator* (vedere la classe *Esempio2* nell'omonimo file Java):

```
…
 DataSource[] dataSources = new DataSource[2];
 dataSources[0] = Manager.createDataSource
                              (audioDevice.getLocator());
 dataSources[1] = Manager.createDataSource
                              (videoDevice.getLocator());
 DataSource dualDataSource =
       Manager.createMergingDataSource(dataSources);
 Player player = Manager.createPlayer(dualDataSource);
 player.start();
…
```

CLONEABLEDATASOURCE

All'interno di un'applicazione possiamo avere più Player operanti ognuno su di un particolare stream dati. In pratica il Player "consuma" i dati che si presentano sullo stream per poterli elaborare, ripresentandoli in uscita in un formato idoneo alla riproduzione. Poiché questi dati in input vengono man mano estratti dal flusso, non rimangono disponibili per eventuali elaborazioni da altri Player. Un caso tipico è quando si vuol salvare su disco le immagini video provenienti da una telecamera digitale e contemporaneamente visualizzarle. La necessità di avere contemporaneamente due operazioni distinte (riproduzione del flusso e salvataggio), costringe l'uso di altrettanti *Player* che debbono coesistere sugli stessi dati. Ma uno dei due esclude l'altro in quanto "consuma" i dati che si presentano.
Per coesistere, ciascun *Player* deve operare su di un proprio flusso di dati, cioé su una copia personale del *DataSource*: come creare e rendere disponibile a ciascuno il proprio stream media? Mediante il metodo statico della classe *Manager*, una versione "clonabile" del *DataSource* originale

```
…
 dataSource=Manager.createCloneableDataSource
                              (dataSource);
…
```

il nuovo *dataSource* mantiene ancora la caratteristica di un *DataSource* ma acquista anche la natura di *SourceClonable*. A questo punto possiamo ottenere da questo oggetto tutte le copie (cloni) che vogliamo, che saranno

indipendenti l'una dall'altra ma non dallo stream originale

```
...
DataSource sourcePlayer1 = ((SourceCloneable)
                               dataSource).createClone();
DataSource sourcePlayer2 = ((SourceCloneable)
                               dataSource).createClone();
Player player1 = Manager.createRealizedPlayer
                                       (sourcePlayer1);
Player player2 = Manager.createRealizedPlayer
                                       (sourcePlayer2);
...
```

Affinchè si abbia un flusso di dati sui cloni, è necessario che anche l'originale sia soggetto ad un flusso. Se provassimo ad avviare i Player a questo stadio, non si avrebbe nulla in riproduzione, in quanto il *DataSource* originale è inattivo.

SALVATAGGIO DI UN MEDIA

A questo punto, per ritornare al caso di riproduzione/salvataggio, ottenuti i cloni necessari si possono predisporre nell'applicazione due moduli. Il primo avrà il compito di controllare e gestire la riproduzione del contenuto audio e video dell'acquisizione, mentre il secondo dovrà gestire il salvataggio dei dati in un file media. Ciascun modulo avrà a disposizione un clone dello stesso *DataSource*. Vediamo brevemente come è possibile salvare un flusso come un file multimediale, ad esempio QUICKTIME.
Per prima cosa dobbiamo procurarci il formato da adottare nella trasformazione del flusso. L'audio dovrà avere un formato con encoding *IMA4* mentre il video può essere *Cinepak* (che sono i formati definiti per il QuickTime).

```
...
Format formats[]=new Format[2];
formats[0]=new AudioFormat(AudioFormat.IMA4);
formats[1]=new VideoFormat(VideoFormat.CINEPAK);
...
```

Poi costruire il descrittore da utilizzare nella definizione del file e il *MediaLocator* da utilizzare

```
...
  FileTypeDescriptor outputType = new
     FileTypeDescriptor(FileTypeDescriptor.QUICKTIME);
  MediaLocator dest = new MediaLocator
                              ("file://./esempio.mov");
...
```

Al posto di un normale *Player*, utilizzeremo un *Processor*: questo perché non dobbiamo riprodurre i dati ma convertirli in un formato compatibile con il file media di memorizzazione. Diversamente dal *Player*, il *Processor* ci consente di accedere al *DataSource* in uscita, cioé ai dati elaborati e convertiti nel formato; per ottenere un oggetto *Processor* in modalità *Realized* abbiamo la necessità di un *ProcessorModel*, costruito sul *DataSource* clonato, sui formati da adottare e sul *FileTypeDescriptor* del file multimediale.

```
...
ProcessorModel processorModel =
  new ProcessorModel(dataSource, formats, outputType)
Player processor = Manager.createRealizedProcessor
                                     (processorModel);
...
```

Costruito il *Processor* si accede al *DataSource* in output e su di esso definiremo il file-writer (in questo caso un *DataSink*). Al termine del flusso originale è necessario chiudere anche il file-writer per completare il salvataggio: non basta quindi bloccare il *Processor* ma è necessario fermare il file-writer, e quindi chiudere il file. La cosa però non è così immediata! Non possiamo fermare il file-writer contestualmente alla chiusura del *Processor*, in quanto possono ancora esserci dei dati sullo stream. Il problema è subito risolto se registriamo un acoltatore che resti in attesa di un evento *EndOfStreamEvent*: appena si raggiunge la fine dei dati presenti sullo stream, questo evento se catturato consentirà di chiudere anche il file-writer rimasto aperto.

```
...
   DataSource source = processor.getDataOutput();
   DataSink filewriter = Manager.createDataSink
                                         (source, dest);
   DataSinkListener listener = new DataSinkAdapter();
   filewriter.addDataSinkListener(listener);
   filewriter.open();
   filewriter.start();
   processor.start();
...
```

Dove la classe *DataSinkAdapter* è:

```
...
class DataSinkAdapter implements DataSinkListener{
        public void dataSinkUpdate(DataSinkEvent event)
        {
          /*Modulo di visualizzazione*/
          if (event instanceof EndOfStreamEvent)
          {try{event.getSourceDataSink().stop();
                    event.getSourceDataSink().close();
           }catch(Exception e){System.out.println
                ("DataSinkAdapter: "+e); System.exit(-1);
        }
            event.getSourceDataSink().close();
      }
    }
  }
...
```

Ing. Antonino Panella

JMFRegistry

L'operazione di mappatura del dispositivo reale con il corrispondente *CaptureDevice* non è completamente automatica ma va controllata attraverso *JMFRegistry*, un'applicazione standalone fornita con la versione 2.1 del tool JMF scaricabile dal sito della SUN. *JMFRegistry* consente di registrare tutti i dispositivi di acquisizione audio e video installati sul PC: la condizione necessaria è che nel sistema siano correttamente presenti tutti i driver necessari per l'identificazione e l'accesso ai dispositivi. Se *JMFRegistry* non riesce a individuare un dispositivo può essere a causa di un errore nei driver o nel processo di installazione.

Come
REALIZZARE UN FTP CLIENT

In questo appuntamento descriveremo gli strumenti che permettono d'implementare le funzionalità di un FTP Client.

Visual Basic

FTP *(File Transfer Protocol)* è il protocollo che definisce le regole per il trasferimento di file tra computer. L'FTP viene usato quando si scaricano, sul proprio computer, file prelevati da Internet, o quando dal proprio computer si trasferiscono file su un altro computer (ad esempio sul Server di un Internet Provider). L'FTP nasce nel 1972 nel MIT *(Massachussets Institute of Technology)*, le specifiche dell'FTP vengono descritte nella RFC 114. L'FTP unisce i comandi del TCP/IP e del Telnet Protocol (Protocolli sviluppati con ARPAnet), si basa sul codice ASCII ed in particolare ne usa, solo, la parte inferiore (ASCII a 7 bit). I comandi dell'FTP oltre a controllare il flusso dei file, da e verso il Server, consentono, quando si trasmettono file, di interagire con il processo in atto sul Server. Ai server FTP è possibile accedere anche attraverso un Web Browser, in questo caso l'indirizzo da specificare è del tipo *"ftp://ftp.microsoft.com"*, naturalmente con questo tipo di richiesta le informazioni visualizzate non saranno delle pagine HTML ma dei nomi di directory e di file. In questo articolo illustreremo come implementare la parte centrale di una FTP Client application. In Visual Basic per tale fine abbiamo vari strumenti: il controllo *Internet Transfer*, le funzioni dell'API di Windows e l'oggetto *Winsock*. Nello spazio di un articolo non sarà possibile presentarli tutti, per questo illustreremo sommariamente il controllo *Internet Transfer*, descriveremo le funzioni dell'API di Windows dedicati ad Internet e faremo solo qualche cenno sul *WinSock Control*.

INTERNET TRANSFER CONTROL

Il controllo *Internet Transfer (INET)* consente di attivare una connessione Internet e di eseguire dei comandi. Il tipo di comando che può essere eseguito dipende dal protocollo di trasmissione selezionato cioè HTTP oppure FTP. I principali metodi del controllo sono: *OpenURL* ed *Execute*.

OpenURL - Questo metodo apre un elemento (pagina, directory) che si trova all'URL specificato. La sintassi è la seguente:

```
object.OpenUrl URL [,datatype]
```

URL è l'indirizzo dove si trova l'elemento che deve essere aperto. *DataType* è un parametro opzionale che specifica il tipo di dato da leggere (*"0" dato stringa, "1" dato come byte array*).

Execute - Questo metodo permette di inviare un comando ad un Server. Naturalmente i comandi che possono essere inviati dipendono dal protocollo selezionato. La sintassi è la seguente:

```
object.Execute url, operation, data, requestHeaders
```

URL, se specificato indica l'URL da utilizzare, nell'*Execute*, altrimenti l'URL utilizzato sarà quello specificato in *OpenUrl*. *Operation* è una stringa che specifica il tipo di operazione da eseguire (tra poco vedremo qualche esempio, non elencheremo tutti i valori, se siete interessati potete controllare la guida in linea di Visual Basic). *Data* è una stringa che specifica le informazioni che verranno utilizzate dall'operazione. *RequestHeaders* è una stringa che specifica informazioni supplementari. Le proprietà principali del controllo sono: *Accetype*, *Protocol*, *URL* ecc. Per esempio *AccessType* è utilizzata per specificare il tipo di accesso alla rete Internet. Come è noto la connessione di un computer client ad Internet può avvenire direttamente o tramite Proxy (cioè attraverso una Intranet collegata alla rete tramite un dispositivo di smistamento) che eventualmente funge da filtro. Per questo i valori che la proprietà può assumere sono:

icNamedProxy cioè accesso tramite proxy;
icDirect accesso diretta ad Internet;
IcUseDefault impostazioni di default, in generale diretto.

Le altre proprietà che utilizzeremo saranno illustrate in seguito. In conclusione aggiungiamo che il controllo INET è in grado di supportare sia la trasmissione sincrona che quella asincrona. La sincrona è associata al metodo *OpenURL*, che durante la trasmissione dei file non permette di eseguire altro codice. L'*asincrona*, invece, è associata al metodo *Execute* che, appunto, consente l'esecuzione di altre operazioni durante il download dei file.

Client server

In una configurazione Client/Server FTP, il Server è un sistema in cui sono archiviati i file (opportunamente organizzati in Directory) che attraverso un FTP Client possono essere manipolati in remoto. Sul Server FTP sono definiti gli account dei Client. I Client dopo aver specificato *Username* e *Password* possono accedere, in lettura/scrittura, ad una parte oppure a tutte (dipende dal tipo di account) le directory del Server. Su alcuni Server, di solito, è definito l'account *anonymous* (con password un indirizzo di posta elettronica o "guest") che contiene le directory pubbliche del sito.

Visual Basic

Tcp/IP

Il protocollo TCP/IP è nato nel mondo Unix e ormai è disponibile su tutti i sistemi operativi.
TCP è un protocollo che garantisce la corretta trasmissione dei dati inviati e ricevuti, conservando la sequenza di trasmissione e ricezione. Insieme all'IP protocol (Internet Protocol), che si occupa essenzialmente dell'instradamento dei pacchetti fino al destinatario, costituiscono il base layer di Internet.
Dunque il TCP/IP è un protocollo orientato alla connessione e concepito per la trasmissione punto-punto.

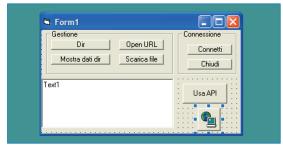



Fig. 1: La form che permette di utilizzare il controllo Internet Transfer.

FTP CLIENT

Iniziamo a descrivere il codice per l'FTP Client.
Create un nuovo progetto Visual Basic con due form (*form1* e *form2*), sulla *form1* inserite un controllo *INET* (naturalmente dopo aver referenziato la libreria *Microsoft Internet Transfer Control*) ed una serie di pulsanti che ci serviranno per abilitare alcune funzionalità del protocollo FTP come: *connessione al Server*, *spostamento e caricamento file* ecc. Inoltre sulla form posizionate un textbox che servirà per mostrare il contenuto dell'URL aperto tramite *OpenUrl* (controllate la Fig. 1).
Analizziamo le istruzioni che consentono di stabilire una connessione e successivamente di utilizzarla per eseguire i comandi FTP. Nel Click del pulsante *Connetti* settiamo i parametri del controllo *INET1* necessari per l'impostazione di una connessione FTP.

```
Private Sub Connetti_Click()
With Inet1
  .UserName = "Anonymous"
  .Password = "guest"
  .AccessType = icDirect
  .RemotePort = 21
  .Protocol = icFTP
  .URL = "ftp://pasta"
  End With
End Sub
```

In particolare effettuiamo una connessione con *UserName "anonimo"* e *Password "guest"* (ospite). Questi valori consentono di accedere alla parte pubblica di uno spazio FTP. Successivamente specifichiamo il tipo di connessione impostando la proprietà *AccessType* su *icDirect*. Specifichiamo che utilizziamo la porta di connessione *21* e il protocollo FTP ed infine impostiamo l'URL del Server cioè *ftp://esempio*. Facciamo notare che il server non è remoto ma si trova sullo stesso computer utilizzato per sviluppare il programma (questo s'intuisce dal modo in cui è specificato l'indirizzo FTP).
Come s'intuisce dal nome del pulsante nel Click di *"Chiudi"* specifichiamo il codice per chiudere la connessione. Questo lo facciamo utilizzando il metodo *Execute* e il comando FTP *"CLOSE"*.

```
Private Sub Chiudi_Click()
  Inet1.Execute , "CLOSE"
End Sub
```

Vediamo ora il comando che ci consente di aprire un file con nome *"provaftptxt.txt"* che si trova nella root dello spazio FTP pubblico (per l'organizzazione delle directory dello spazio FTP controllate la Fig. 3).

```
Private Sub Openurl_Click()
  Text1.Text = Inet1.openurl("esempio/provaftptxt.txt")
End Sub
```

Nel *Click* del pulsante *Openurl* richiamiamo il metodo *OpenUrl* passandogli l'indirizzo ed il nome del file da aprire, il risultato di questa richiesta verrà restituito nel *textbox Text1*. Per scaricare il file localmente utilizziamo la seguente procedura:

```
Private Sub Scarica_Click()
  Inet1.Execute , "GET provaftptxt.txt c:\provadowload2.txt"
End Sub
```

In cui dopo aver invocato il metodo *Execute* sull'oggetto *Inet1* gli passiamo come operazione FTP una *"GET"* (cioè preleva) che ha come parametri il file da prelevare ed il percorso locale dove trasferire il file (cioè *c:\provadowload2.txt*). Come accennato la suddetta operazione avviene in modalità asincrona, quindi la stessa operazione poteva essere effettuata in modo sincrono utilizzando il metodo *OpenURL*. La lista dei file della root la richiediamo attraverso il metodo *EXECUTE* e l'Operatore *DIR*. Se però volessimo la lista dei file presenti in una sotto directory, ad *Execute* dovremmo passare *"DIR"* più il nome della sotto directory nella forma: *DIR [nome]*, ecc. Quindi possiamo considerare un'istruzione del tipo:

```
Private Sub dir_Click()
  Inet1.Execute , "DIR"
End Sub
```

Naturalmente, la creazione della connessione deve essere fatta prima dell'esecuzione delle altre operazioni per questo conviene inserire del codice per il controllo di questo aspetto (questo lo faremo quando descriveremo le funzioni dell'API). Notate come in Fig. 1, la *Form1* ospita un pulsante con nome *UsaAPI*, questo serve per mostrare la *form2* dove abbiamo previsto il codice che richiama alcune funzioni dell'API. Iniziamo, quindi, la descrizione della seconda parte del nostro progetto che prevede l'uso delle API per realizzare le operazioni che abbiamo implementato in precedenza attraverso il controllo *INET*.

FTP CLIENT CON LE API

Anche sulla *form2* posizioniamo una serie di pulsanti (Fig. 2). Le funzioni API le dichiariamo in un modulo (*Module1.bas*). In esso dichiariamo anche le seguenti costanti e variabili.

```
Public Const INTERNET_SERVICE_FTP = 1
```

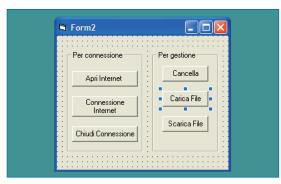



Fig. 2: **La form che permette di utilizzare le funzioni dell'API.**

```
Public Const INTERNET_SERVICE_GOPHER = 2
Public Const INTERNET_SERVICE_HTTP = 3
Public sessioneinternet As Long
Public sessioneftp As Long
`qui vanno inserite le dichiarazioni delle funzioni API
`che descriveremo tra poco
```

Sessioneinternet e *sessioneftp* servono per conservare l'handle della sessione Internet e della sessione ftp; le costanti verranno utilizzate dalla funzione *InternetConnect* per specificare il tipo di servizio (come sarà chiaro tra poco).
Nel *Click* del pulsante *Apri Internet* creiamo la sessione Internet.

```
Private Sub ApriInternet_Click()
  sessioneinternet = InternetOpen("progetto1", 0,
                    vbNullString, vbNullString, 0)
End Sub
```

La sessione internet viene inizializzata richiamando la funzione API *InternetOpen* di cui vediamo brevemente la sintassi:

```
Public Declare Function InternetOpen
        Lib "wininet.dll" Alias "InternetOpenA" (
                ByVal sAgent As String,
                ByVal nAccessType As Long,
                ByVal sProxyName As String,
                ByVal sProxyBypass As String,
                ByVal nFlags As Long) As Long
```

Dove:

- **sAgent**, è il nome dell'applicazione che chiama la funzione API;
- **nAccessType**, specifica il tipo di accesso (diretto, tramite proxy, ecc.);
- **sProxyName**, è una stringa che contiene il nome dell'eventuale proxy, può essere *null*;
- **sProxyBypass**, nome del proxy opzionale, può essere *null*;
- **nFlags**, flag indicante diverse opzioni: connessione asincrona, da cache, offline, per questo parametro abbiamo utilizzato il valore di default.

La funzione restituisce l'handle della connessione che verrà utilizzata nella procedura *ConnessioneInternet*. Notate che le API si trovano nel file *WinInet.dll*.

```
Private Sub ConnessioneInternet_Click()
 If sessioneinternet = 0 Then
  MsgBox "bisogna prima creare una sessione Internet"
  Exit sub
End If
  sessioneftp=InternetConnect(sessioneinternet, "pasta", "21"
        "anonymous", "guest", INTERNET_SERVICE_FTP, 0, 0)
End Sub
```

Nella suddetta procedura testiamo innanzitutto l'esistenza di una sessione Internet, se la risposta è affermativa andiamo ad "aprire" una sessione ftp richiamando la funzione API *InternetConnect*, della quale vediamo sommariamente la sintassi.

```
Public Declare Function InternetConnect
        Lib "wininet.dll" Alias "InternetConnectA" (
                ByVal hInternetSession As Long,
                ByVal sServerName As String,
                ByVal nServerPort As Integer,
                ByVal sUserName As String,
                ByVal sPassword As String,
                ByVal nService As Long,
                ByVal dwFlags As Long,
                ByVal dwContext As Long) As Long
```

- **hInternetSession**, handle sessione internet (creata come descritto in precedenza);
- **sServerName**, stringa che indica il nome del server;
- **nServerPort**, porta del server, per il servizio FTP (sarà la 21);
- **sUserName**, username per la connessione;
- **sPassword**, password per la connessione;
- **nService**, tipo di servizio richiesto (FTP, http).

Gli ultimi due parametri non li abbiamo utilizzati (l'impostiamo sul valore zero). Questa funzione restituisce l'handle della sessione FTP (se l'operazione ha successo). Per chiudere la connessione utilizziamo il codice inserito nel pulsante *Chiudi Connessione*.

```
Private Sub ChiudiConnessione_Click()
  Call InternetCloseHandle(sessioneinternet)
  Call InternetCloseHandle(sessioneftp)
End Sub
```

Con la *ChiudiConnessione_Click* chiudiamo sia la sessione Internet che quella ftp, il tutto attraverso *InternetCloseHandle* che restituisce il valore booleano *true* se l'operazione ha avuto esito positivo. La sintassi di questa funzione è semplice:

```
Public Declare Function InternetCloseHandle
        Lib "wininet.dll" (ByVal hInet As Long) As Integer
```

Visual Basic

Web server

Gli esempi fatti nell'articolo sono eseguiti utilizzando Visual Basic e IIS (*Internet Information Server*) che come è noto permette di gestire un Server Web ed un Server FTP.
Naturalmente con i sistemi operativi Windows 2000 e Windows XP si lavora con IIS, mentre con Windows 95/98 si lavora con *Personal Web Server* (PWE).
Quest'ultimo si trova nella suite Visual Studio oppure nella cartella *add-ons* del Cd del sistema operativo.
Se sul vostro computer non avete installato né IIS e né PWS potete testare gli esempi iscrivendovi ad una Community come Lycos:
http://www.tripod.lycos.it/

Ultimo avvertimento il PWS o l'IIS installato deve essere nella stessa lingua di Visual Basic altrimenti avrete delle sgradite sorprese.

Visual Basic

Winsock

Winsock è il controllo che consente di utilizzare i protocolli di trasmissione TCP/IP, UDP, FTP ecc. Il controllo WinSock, fa parte della libreria Microsoft Winsock Control 6.0 (in cui è presente l'ocx MSWINSCK). Questo controllo è stato descritto nell'articolo: Il controllo per accedere ai servizi di rete, nel quale abbiamo illustrato come implementare una Chat.

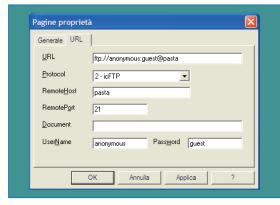



Fig. 3: La finestra delle proprietà del controllo Internet Transfer

Accetta come parametro l'handle della connessione. Ora vediamo come effettuare il Download di un file. Nel pulsante Scarica file inseriamo il seguente codice:

```
Private Sub ScaricaFile_Click()
 If sessioneftp = 0 Then
   MsgBox "bisogna prima creare una sessione ftp"
   Exit sub
 End If
 If FtpGetFile(sessioneftp, "provaftptxt.txt",
        "c:\provadowload.txt", False, 0, 1, 0) = False Then
  MsgBox "errore"
 End If
End Sub
```

Dopo aver constatato l'esistenza di una sessione ftp, la funzione richiama l'API che consente di scaricare il file e cioè la *FtpGetFile*, di cui descriviamo i parametri fondamentali:

```
Public Declare Function FtpGetFile
   Lib "wininet.dll" Alias "FtpGetFileA" (
                              ByVal hFtpSession As Long,
                         ByVal lpszRemoteFile As String,
                             ByVal lpszNewFile As String,
                           ByVal fFailIfExists As Boolean,
                    ByVal dwFlagsAndAttributes As Long,
                                   ByVal dwFlags As Long,
                   ByVal dwContext As Long) As Boolean
```

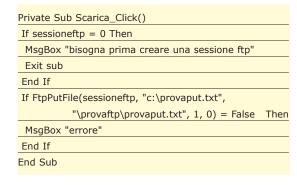
- **hFtpSession**, handle della sessione ftp;
- **lpszRemoteFile**, file da scaricare;
- **lpszNewFile**, percorso e nome del file destinatario.

Per gli altri parametri utilizziamo i valori di default. Notate che il file da scaricare è *provaftptxt.txt* e che verrà scaricato nella root del disco *c:/* (con il nome *provadowload.txt*). Vediamo come effettuare un upload verso il server "remoto", per tale scopo utilizziamo la seguente funzione.

```
Public Declare Function FtpPutFile
                       Lib "wininet.dll" Alias "FtpPutFileA" (
                                  ByVal hFtpSession As Long,
                              ByVal lpszLocalFile As String,
                             ByVal lpszRemoteFile As String,
                                      ByVal dwFlags As Long,
                       ByVal dwContext As Long) As Boolean
```

- **hFtpSession**, handle della sessione ftp;
- **lpszLocalFile**, file da scaricare sul server remoto;
- **lpszRemoteFile**, percorso e nome del file sul server remoto;
- **dwFlags**, flag indicante il tipo di trasferimento (ASCII o binario);
- **dwContext**, impostato su zero.

Ecco come utilizziamo questa funzione:

```
Private Sub Scarica_Click()
 If sessioneftp = 0 Then
  MsgBox "bisogna prima creare una sessione ftp"
  Exit sub
 End If
 If FtpPutFile(sessioneftp, "c:\provaput.txt",
              "\provaftp\provaput.txt", 1, 0) = False   Then
  MsgBox "errore"
 End If
End Sub
```

La funzione restituisce *true* se l'operazione avviene correttamente. L'eliminazione di un file sul server può essere fatta attraverso la funzione *FtpDeleteFile()*.

```
 FtpDeleteFile(sessioneftp, "\provaftp\provaput.txt")
```

In questo caso i parametri sono semplicemente: l'handle della sessione ftp ed il nome del file da cancellare sul server. Dalla funzione sarà restituito il valore *true* o *false* in base all'esito dell'operazione.

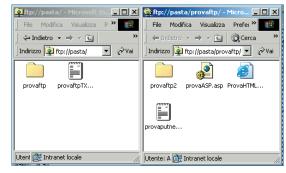



Fig. 4: Uno spazio FTP visto attraverso il Browser Internet Explorer Internet Transfer.

CONCLUSIONE

Ora che avete capito come utilizzare il protocollo FTP potete realizzare un FTP Client con l'interfaccia nello stile Explorer. Questo tipo d'interfaccia di solito prevede un "albero" (treeview) che presenta la struttura delle directory presenti nell'account ed una listview che mostra i file contenuti nella directory selezionata.

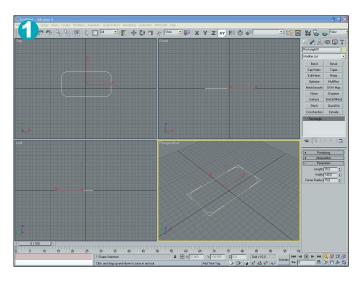
Massimo Autiero

Modellazione tramite Surface

▶ 3D Studio Max

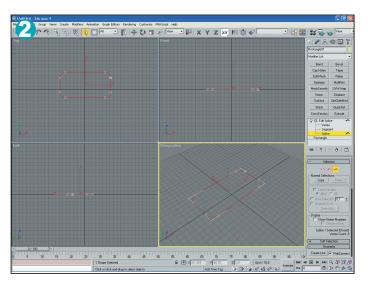
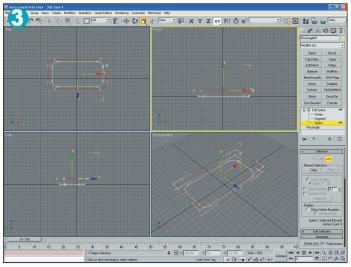
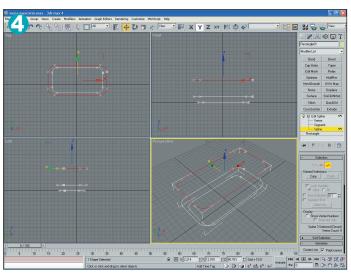
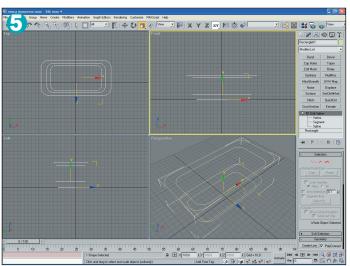
Pietro Canini

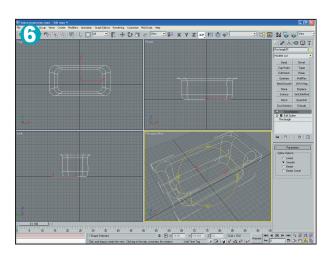
Vediamo come utilizzare uno tra i più versatili strumenti di modellazione, il Surface, costruendo una vasca. Non mancheranno gli strumenti già visti in precedenza, come: le operazioni booleane, le estrusioni ed alcuni modificatori.

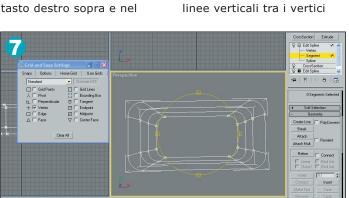
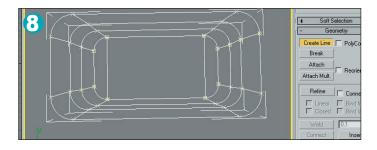
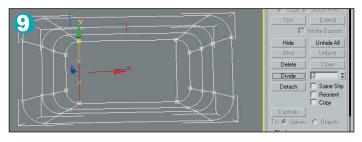
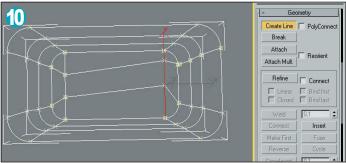
Ben ritrovati davanti ai nostri PC con 3dsmax (in uscita la versione 5.0, con tante novità; prima tra tutte un nuovo motore di rendering cui è possibile applicare *Radiosity* e *Global Illumination*). Uno dei metodi migliori per modellare oggetti curvi, organici e non, complessi sono gli strumenti di Superficie. Questi strumenti sono composti da due modificatori: *Cross Section* e *Surface*. Il *Cross Section* crea delle spline di congiunzione tra spline esistenti. Ad esempio se abbiamo un cerchio sotto ed un cerchio sopra e gli applichiamo il *Cross Section*, vediamo delle spline che collegano i due cerchi per formare un cilindro. Nota bene che le spline che creeremo noi, dovranno essere ordinate e unite (se le creiamo separatamente) col comando *Attach*. Applicando poi il modificatore *Surface* al modello spline, avremo delle patch che ricoprono la struttura da noi creata. Nel caso del cilindro, oltre al reticolato in wireframe, avremo anche le patch che formano una struttura chiusa e visibile. Dei consigli utili per chi usa questi strumenti sono: l'utilizzo di figure di riferimento da applicare su dei piani in background; lavorazione su metà modello, per poi specchiarlo (se è simmetrico); utilizzare lo *snap3d* a vertice; creare una copia del modello usando l'opzione *Reference* ed applicandogli il modificatore *Surface*, in modo da vedere, mentre si modellano le spline, i cambiamenti sull'altro oggetto; la maggior parte delle volte, quando dei vertici coincidono, Max chiederà di saldare i vertici, è opportuno dire di *No* in quanto è sempre possibile unirli dopo col comando *Weld*. Con questo metodo dunque, è possibile creare teste umane, automobili, aerei, ecc…
Modelleremo un oggetto semplice nella sua creazione, ma fondamentale per capire subito questo metodo di modellazione, la vasca. Vedremo, dopo la modellazione: il materiale da applicargli e cioè un bianco con leggere riflessioni sfocate tramite lo shader Raytrace; altri parametri quali align, array, ed alcuni modificatori. Iniziamo a modellare…

◀1 La prima spline

Avviamo 3DSMax ed iniziamo a creare le spline necessarie per il nostro **Surface**. Andate quindi nel pannello **Create/ Shapes** e cliccate su **Rectangle**. Nella vista **Top** create un rettangolo di dimensione qualsiasi, ed andiamo nel pannello **Modify** per definirne la grandezza. Nei parametri del rettangolo **(Rullout Parameters)** impostiamo **Length = 70; Width = 140; Corner Radius = 15.** Questa sarà la base della nostra vasca, ora creiamo le altre spline partendo da questa.

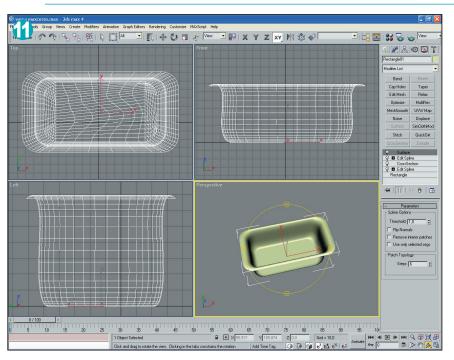
2-3-4-5 Le altre spline

Sempre nel pannello **Modify** aprite la tendina dei modificatori e dalla lista scegliete **Edit Spline**, in modo da trasformare il rettangolo in una forma (**Spline**) modificabile (**Edit**). Ora col modificatore applicato scegliete di lavorare a livello **Spline**, quindi selezionate il rettangolo che diventerà rosso. Nella vista **Front** spostate (**Select and Move**), solo sull'asse **Y**, il rettangolo in alto (di circa 15 unità) tenendo premuto **Shift** da tastiera (in modo da clonarlo). Ingranditelo (**Select and Uniform Scale**) di circa 15 unita (3DStudio in **X, Y, Z** segnerà quindi 115 unità perché parte da 100). Ora ancora col **Select and Move** e **Shift** premuto cloniamo sempre sull'asse **Y** l'ultima spline creata, spostandola in alto di circa 45 unità. Stesso procedimento per quest'ultima spline creata, questa volta su di 6 unità. Ora ingranditela di 20 unità (**X, Y, Z 120 u.**).

Il Cross Section 6

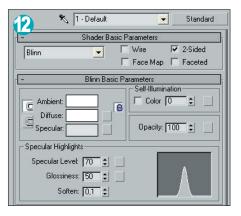
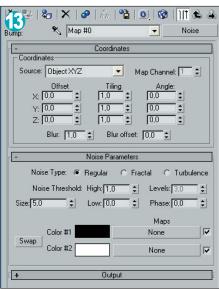
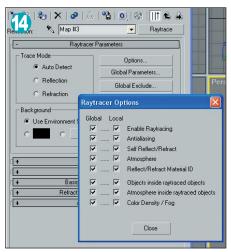
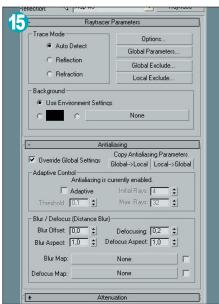
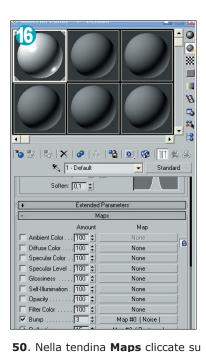
A questo punto disattivate la modalità spline (ricliccandoci sopra). Dalla tendina dei modificatori scegliete **Cross Section**. Nei suoi parametri spuntate **Smooth** in modo da avere le linee di giunzione smussate. Se provate ad applicargli anche il modificatore **Surface**, noterete che la forma della vasca c'è, ma non è chiusa alla base. Quindi rilevate il **Surface** che avete appena messo per provare, lasciando **Cross Section** come ultimo modificatore. Nella lista dei modificatori scegliete di nuovo **Edit Spline**.

Chiusura 7-8-9-10

Scegliete di lavorare a livello **Segment**. Mettete la vista prospettica in modo da poter vedere tutti i vertici che formano la base (la prima spline creata). Attivate lo **Snap3D**, cliccateci col tasto destro sopra e nel **Grid and Snap Settings** spuntate solo l'opzione **Vertex**. Nel rullout **Geometry**, cliccate su **Create Line** e disattivate l'opzione accanto **PolyConnect**. Nella vista prospettica, tracciate delle linee verticali tra i vertici

superiori e quelli inferiori (con lo snap attivo, la freccetta punterà direttamente sui vertici). Disattivate il **Create Line** ricliccandoci sopra. Ora selezionate una linea verticale che avete appena creato e sempre nel rullout **Geometry** impostate il valore **2** accanto a **Divide**, quindi cliccate su **Divide**.

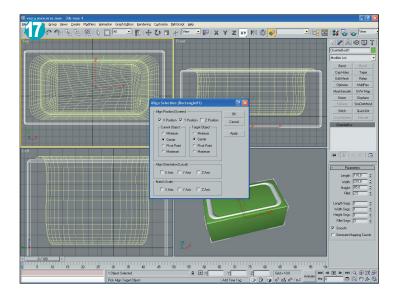
Ora al segmento verranno aggiunti 2 vertici; fate lo stesso per l'altra linea verticale. Riattivate **Create Line** e create le linee orizzontali come in Fig. 10 (ad ogni linea creata, quindi da vertice a vertice, cliccate col tasto destro del mouse per interrompere il segmento).

11 Il Surface

Disattivate **Create Line** e la modalità di lavorazione sul segmento, ricliccando su **Segment**. Dalla lista dei modificatori scegliete **Surface** e lasciate i parametri così come sono. Ecco creato l'interno della vasca. Se dovessero esserci delle parti bucate sulla base, significa che qualche vertice non è sovrapposto a quello sottostante; quindi spostatelo esattamente sopra e quando si chiederà di saldare, scegliete **No**. Abbiamo visto quindi che il modificatore **Surface** funziona su poligoni quadrati, quindi quando sono composti da quattro vertici. Infatti il procedimento fatto per chiudere la base non è altro che l'aggiunta di vertici affinché si formino dei poligoni quadrati. Continuiamo con la creazione della vasca.

12-13-14-15-16 Il materiale

Come avrete notato, un lato della vasca è trasparente; quindi aprite il **Material Editor** e rimediamo a questo fatto. In uno **Slot** vuoto, nella tendina **Shader Basic Parameters** spuntate l'opzione **2-Sided** (in modo da rendere visibile la parte trasparente); impostate il colore **Diffuse Bianco** (**RGB = 255**); **Specular Level = 70**; **Glossiness =**

50. Nella tendina **Maps** cliccate su **None** accanto a **Bump** per definire un leggerissimo rilievo; dalla lista scegliete **Noise**. Impostate, nei parametri del **Noise**, **Size = 5**. Tornate un livello indietro (**Go to Parent**) e mettete **Bump Amount = 3**. Ora cliccate su **None** accanto a **Reflection** e dalla lista scegliete **Raytrace**. In **RayTrace Parameters** cliccate su **Option** ed attivate l'opzione **Antialiasing** in **Global**. Ora aprite il menù a tendina **Antialiasing** e puntate l'opzione **Override Global Settings**; in **Defocusing** impostate 0,2.
In questo modo avrete delle riflessioni sfocate ma con un notevole aumento di tempo nel rendering. Tornate un livello indietro e mettere **Reflection Amount = 15**. Applicate il materiale alla vasca se non l'avete ancora fatto.

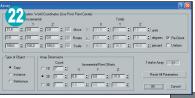
La base 17

Creiamo, tramite le operazioni booleane, la struttura base della vasca.
Andate nel pannello **Create / Geometry / Extended Primitives** e cliccate su **Chamfer Box**. Nella vista **Top** create la geometria di dimensioni **Length = 110; Width = 210; Height = 85; Fillet = 2,5.**
Con il box selezionato, cliccate sul comando **Align** (si può trovare anche nel menù a tendina in alto **Tools / Align**; oppure **ALT+A**) e poi sulla vasca.
Nei parametri di allineamento in **Align Position** spuntate l'opzione **X Position** e **Y Position**; quindi cliccate su **Apply** e poi su **Ok**.
Ora il box risulterà centrato rispetto alla vasca.

18-19-20 La sottrazione

Nel pannello **Create/ Geometry / Standard Primitives** cliccate su **Box** e nella vista **Top** create la geometria di dimensioni **Length = 87,048; Width = 169,492; Height = 86,629** e posizionatelo come in Fig. 18. Selezionate il **Chamfer Box** creato prima, andate nel pannello **Create / Geometry /** Compound **Objects / Boolean** e cliccate su **Pick Operand B** e selezionate l'ultimo **Box** creato. Lasciando i parametri booleani così come sono, avrete una sottrazione del box, l'interno della vasca diverrà di nuovo visibile.

21-22-23 Le maioliche

Creiamo un **Chamfer Box** di dimensioni **Length = 30; Width = 30; Height = 2,5; Fillet = 0,5** e posizionatelo come in Fig. 21; ecco la nostra prima maiolica che moltiplicheremo per creare le altre, solo sul lato frontale (se le volete anche sugli altri lati, seguite di nuovo questo procedimento).

Con la maiolica selezionata scegliete l'opzione **Array** (menù a tendina **Tools/Array**; oppure l'icona nella barra degli strumenti in alto). Mettete: **Incremental / Move / X = 31; Array Dimension** spuntate l'opzione **2D**; **Count 1D = 7; Count 2D = 3; Incremental Row Offset Z = -31**, quindi cliccate su Ok.

24-25-26 Aggiustamenti

Noterete che le maioliche a destra e in basso hanno dimensioni non conformi, andiamo ad aggiustarle. Per far sembrare che sono state tagliare dalla mano dell'uomo, selezionate tutta la fila in basso, andate nel pannello **Modify** e dalla lista dei modificatori scegliete **Slice**. Nelle sue opzioni sceglieste **Remove Bottom**. Si creerà così un piano di ritaglio che elimina la parte sottostante. Bisogna però regolare il piano per far si che tagli le maioliche quando inizia lo smussamento del **chamfer box** sottostante. Fate quindi click su **Slice** in modo da lavorare a livello sotto l'oggetto e quindi muovere il ripiano (**Gizmo**) sull'asse **Y**. Una volta spostato il gizmo ricliccate su **Slice** per disattivare la modalità **sub-objects**; selezionate la fila di maioliche a destra ed eseguite lo stesso procedimento, questa volta però ruotando il gizmo di 90 gradi. Ora renderizzate.

Conclusioni

Il tutorial si conclude qua, potete perfezionare la scena applicando texture agli oggetti creati; magari sulle maioliche applicando sempre una leggera riflessione sfocata. Abbiamo visto come il nuovo metodo di modellazione Surface sia efficiente e se usato con maestria, perdendoci molto più tempo, si possono tirare fuori delle superfici organiche realistiche.
Il mio spazio si conclude qui, quindi ci ritroviamo i prossimi mesi con un altro interessante articolo.

Simulare
UNA RETE CON NETWORK SIMULATOR

Networking

Analizzare il comportamento di una rete, locale, metropolitana, cablata, wireless e di qualsiasi altra natura. È fondamentale per ricavare le massime prestazioni.

Network Simulator, brevemente NS, è un simulatore universale di reti, conosciutissimo in ambito accademico, ma non solo, permette di progettare una rete di telecomunicazioni di qualsiasi tipo e ottenere informazioni di ogni genere sul suo funzionamento. NS permette di simulare praticamente ogni aspetto della rete, partendo dai suoi componenti fondamentali, cioè nodi, router, link, passando per i diversi protocolli coperti, ad esempio CSMA/CD a livello di linea, IP a livello di rete, TCP a livello di trasporto e così via, coprendo praticamente l'intera pila OSI. Il cuore di NS è scritto interamente in C++ ed in filosofia open source, permettendo di creare nuovi oggetti o modificare quelli esistenti. Infatti, col susseguirsi delle sue varie release, attualmente siamo giunti alla versione 2.1b9, sono state varie le aggiunte e modifiche da parte dei gruppi di lavoro sparsi su tutto il pianeta. L'utente normale, il sistemista di rete, lo studente di reti di telecomunicazioni, ha inoltre a disposizione un linguaggio di scripting, OTcl *(Object Tool Command Language)*, che in maniera semplice e veloce permette di descrivere una topologia di rete, il suo traffico, il movimento dei nodi mobili, e i risultati che si vogliono ricavare dalla simulazione.

PERCHÉ DUE LINGUAGGI?

L'Object Tcl, è un'estensione del Tcl/Tk per la programmazione object-oriented, in pratica fornisce un'interfaccia per l'utente, che così non deve assolutamente mettere mano al codice C++. In pratica NS è costituito da una gerarchia di classi C++, detta gerarchia compilata, ed una gerarchia OTcl detta invece interpretata. Le due gerarchie sono strettamente legate, dal punto di vista dell'utente c'è una relazione uno ad uno fra una classe C++ ed una OTcl. L'utente crea nuovi oggetti attraverso l'interprete, questi oggetti vengono istanziati al suo interno e a questo punto vengono cercati i corrispondenti oggetti nella gerarchia compilata. Sembra più complicato a dirsi che a farsi, ma c'è naturalmente un motivo che porta all'uso di due linguaggi diversi per un simulatore di reti. Da una parte la simulazione dettagliata di protocolli di rete richiede un linguaggio di programmazione di sistema, che sia efficiente nella manipolazione di byte, degli header dei pacchetti, e nell'esecuzione di algoritmi su grosse quantità di dati. Per questi compiti quello che importa maggiormente è la velocità di esecuzione run-time. Dall'altra parte invece simulare una rete o progettarla, richiede la possibilità di variare parametri, creare gli scenari di simulazione, rimandare in esecuzione una stessa simulazione più volte, compiti per cui una veloce esecuzione run-time è meno importante rispetto alla velocità di iterazione, cioè cambiamento del modello – riesecuzione.

COMINCIAMO

Esistono varie distribuzioni del simulatore, per Unix e windows, da compilare a partire dai vari componenti che lo costituiscono, o in pacchetti all-in-one che contengono tutto ciò che ci serve. Consigliamo di usare NS sotto linux, sul quale d'altronde è stato sviluppato, perché avremo a disposizione una marea di tool complementari che ci faciliteranno il compito di simulazione ed analisi dei risultati, oltre a poter dare un'occhiata all'intero codice sorgente di NS. Il pacchetto all-in-one per le piattaforme Unix è pero, nell'ultima release, di quasi 40Mb, così chi vuole iniziare a sperimentare con Ns può anche procurarsi le versioni precompilate per Windows, sia di Ns che del tool Nam. Nam, o Network Animator, darà un aspetto anche grafico alle nostre simulazioni, permettendoci di seguire visivamente la loro evoluzione, ad esempio seguendo il movimento dei pacchetti lungo i link, il riempirsi delle code in un router o anche il movimento di nodi mobili. Nei pacchetti all-in-one sono inoltre contenuti tutti i sorgenti sia del simulatore NS, che di Nam, che di tutti gli altri tools. Ad esempio per facilitare la creazione dei grafici dei risultati, troviamo l'utility Xgraph, mentre per facilitare la creazione degli scenari delle simulazioni, e per poter così programmare intere campagne di simulazione in modo da avere risultati più reali, troviamo degli script e dei file eseguibili per la generazione del traffico e per la generazione dei movimenti dei nodi mobili. Senza

File sul CD
\soft\codice\NSimulator\

Prestazioni

Il C++ è veloce da eseguire ma lento da modificare, rendendolo ideale per l'implementazione dettagliata di un protocollo. L'OTcl gira più lentamente ma può essere cambiato velocemente e interattivamente.

Networking

contare il fatto che possiamo anche dare un'occhiata ad una grossa raccolta di esempi su ogni tipo di rete, cosa che senz'altro aiuta più di ogni manuale. Sia in Windows che in Unix si lavora in ambiente a riga di comando. L'input da dare al simulatore è costituito da un file di testo con estensione *tcl* contenente non solo la descrizione della rete da simulare ma anche le direttive del simulatore, ad esempio il tempo di simulazione, i file di input/output, gli eventi da tracciare, o anche dettagli puramente grafici come il colore e la dimensione dei nodi.

UN PRIMO ESEMPIO

Vediamo un primo semplice esempio per cominciare a prendere pratica con il linguaggio OTcl e i tool che abbiamo a disposizione. Supponiamo di voler simulare una semplice rete costituita da due nodi, connessi da un link full-duplex, e l'unico tipo di traffico presente sia di tipo costante, con pacchetti di dimensione 500 bytes spediti da un nodo all'altro. Iniziamo a scrivere il nostro primo script, con un semplice editor di testo. Innanzitutto si definisce un oggetto *Simulator* ed il nome del file di output per *Nam*:

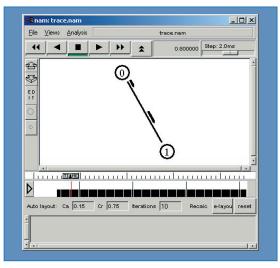



Fig. 1: **Il risultato della nostra prima simulazione.**

```
set ns [new Simulator]
set namFile [open trace.nam w]
```

poi, con il seguente commando, diciamo ad Ns di tracciare tutto sul file *.nam*

```
$ns namtrace-all $namFile
```

A questo punto dobbiamo definire i due nodi ed il link che li congiunge, con una capacità di 1Mb e tempo di propagazione di 20ms:

```
set Node0 [$ns node]
set Node1 [$ns node]
$ns duplex-link $Node0 $Node1 1Mb 20ms DropTail
```

```
set DuplexLink0 [$ns link $Node0 $Node1]
```

Non resta che generare il traffico, dal nodo che abbiamo chiamato *Node0* faremo partire dei pacchetti da 500 byte a bit-rate costante di uno ogni 0.005 secondi, che verranno ricevuti da *Node1*. Ecco come fare nel dettaglio: gli end-point di rete dove vengono generati e consumati pacchetti sono detti Agent, per prima cosa definiamo un *Agent* di tipo UDP, chiamandolo *UDP0*:

```
set UDP0 [new Agent/UDP]
```

e lo assegniamo al nodo che abbiamo chiamato *Node0*:

```
$ns attach-agent $Node0 $UDP0
```

Quindi abbiamo bisogno di un *Agent* che riceva i pacchetti, per fare ciò creiamo un *Agent* di tipo *Null*, e lo "attacchiamo" al secondo nodo:

```
set Null0 [new Agent/Null]
$ns attach-agent $Node1 $Null0
```

All'Agent UDP del nodo *Node0* assegniamo inoltre il generatore di traffico *Constant Bit Rate*, con i relativi parametri:

```
set CBR0 [new Application/Traffic/CBR]
$CBR0 set interval_ 0.005
$CBR0 set packetSize_ 500
$CBR0 set random_ 0
$CBR0 attach-agent $UDP0
```

Ora possiamo conettere i due *Agent*:

```
$ns connect $UDP0 $Null0
```

Essendo Ns un simulatore ad eventi discreti, bisogna decidere il tempo di durata della simulazione, che sceglieremo pari a 10 secondi e gli istanti di inizio e fine della trasmissione dei pacchetti. Per comodità inoltre definiamo una semplice procedura che ci servirà a chiudere il file di output e lanciare l'esecuzione di Nam.

```
proc finish {} {
 global ns
 global namFile
 $ns flush-trace
 close $namFile
 exec nam trace.nam &
 exit 0 }
```

Il generatore di traffico CBR inizierà a trasmettere all'istante 0.5 secondi e finirà all'istante 4.5 secondi:

```
$ns at .5 "$CBR0 start"
$ns at 4.5 "$CBR0 stop"
$ns at 10.0 "finish"
```

Linux

Nell'articolo abbiamo visto e dimostrato come sia possibile con NS simulare una qualsiasi tipologia e topologia di rete, sia come punto di partenza per il progetto, sia come strumento per l'analisi delle prestazioni. Per chi è davvero interessato consigliamo di scaricare e installare la versione Linux di Ns e naturalmente la completissima documentazione.

Networking

Infine possiamo chiamare il metodo *run* del simulatore:

```
$ns run
```

Supponendo di aver salvato lo script come *esempio1.tcl* possiamo far partire la simulazione dal prompt dei comandi con: > *ns esempio1.tcl*. Dopo qualche secondo, per una simulazione così semplice, vedremo apparire la finestra del nam (vedi Fig. 1). Ns permette naturalmente di simulare situazioni molto più dinamiche, come tutti sappiamo è purtroppo possibile che un link fra due nodi di una rete ad un certo punto si guasti o si congestioni, o comunque non permetta la trasmissione di pacchetti su di esso. La caduta di un link è simulabile in Ns in modo molto semplice. Ad esempio se nel nostro primo esempio vogliamo che il link fra i due nodi ad un certo punto cada, diciamo nell'intervallo di tempo [2,3] secondi, basta aggiungere due righe nella seguente maniera:

```
$ns rtmodel-at 2.0 down $Node1 $Node2
$ns rtmodel-at 3.0 up $Node1 $Node2
```

Anche nella finestra del Nam vedremo questa situazione, con il link che cambia colore e i pacchetti che vngono persi.

SIMULIAMO UNA RETE WIRELESS MOBILE

Cerchiamo di esplorare le capacità di NS con un esempio più complesso, ma che, con gli strumenti a disposizione, non richiede molto più di quello che abbiamo visto. Supponiamo di voler simulare una rete wireless 802.11 con i nodi aventi capacità di movimento, cioè abbiamo a che fare con una vera e propria rete mobile. Inoltre vogliamo ottenere dalla simulazione risultati riguardanti ad esempio quanti dei pacchetti spediti arrivano a destinazione, cioè il cosiddetto "Packet Delivery Ratio". La prima cosa da fare è quella di configurare i nodi mobili, compresa l'antenna e l'interfaccia radio. Usiamo un antenna omnidirezionale, e settiamo i parametri in modo da averla ad un altezza di 1,5 metri e con guadagno unitario sia in trasmissione che in ricezione:

```
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0
```

Per quanto riguarda l'interfaccia radio, la configuriamo in modo che funzioni come una *Lucent WaveLan 914Mhz.*, basta consultare il manuale della scheda e i parametri corrispondenti sono:

```
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
```

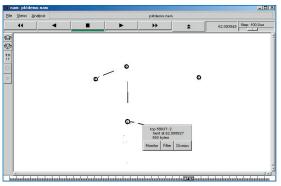



Fig. 2: Un clic su un pacchetto per scoprirne tutte le caratteristiche.

```
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
```

A questo punto configuriamo i nodi mobili, basta variare i parametri del comando *node-config*, descritti in modo esauriente nel manuale ufficiale di Ns, quelli che ci interessano maggiormente sono i seguenti:

```
node-config -adhocRouting DSR \  #protocollo di routing ad hoc
-llType LL \ #tipo di link layer
-macType Mac/802.11 \  #protocollo di livello MAC
-ifqType Queue/DropTail/PriQueue \#accoramento
                                   e droppaggio dei pacchetti
-ifqLen 50 \ #lunghezza della coda
-antType Antenna/OmniAntenna \          #tipo di antenna
-propType Propagation/TwoRayGround \ #propagazione
                                              del segnale
-phyType Phy/WirelessPhy \              #livello fisico
-channelType $opt(chan) \               #canale fisico
-topoInstance $wtopo \ #la topografia della rete
-agentTrace OFF \
-routerTrace ON \ #tracciamo solo i pacchetti a livello
                                                   Router
-macTrace OFF
```

Con altri parametri, qui non indicati, è anche possibile definire il modello energetico del nodo, ad esempio la potenza di trasmissione e ricezione, oltre ad altre caratteristiche che qui stiamo trascurando. Ora può avvenire la creazione vera e propria dei nodi, supponiamo di volerne 5:

```
for {set i 0} {$i < 5 } {incr i}
{
  set node_($i) [$ns_ node]
  $node_($i) random-motion 0 ;# 0 disabilita
                                  il movimento casuale}
```

Per quanto riguarda la mobilità dei nodi, sotto Linux, è comodo e conveniente utilizzare l'utility *setdest*, che crea dei pattern di movimento casuali su file, direttamente richiamabili dallo script, specificando il nome del file da includere con il comando:

Sul Web

- **Sito ufficiale**
  http://www.isi.edu/nsnam/ns/index.html
- **NS by example**
  http://nile.wpi.edu/NS/
- **Marc Grei's Tutorial**
  http://www.isi.edu/nsnam/ns/tutorial/index.html

Networking

```
source miadir/file_traffico
```

Altrimenti possiamo definire da codice il modello di movimento con delle righe come le seguenti:

```
# inizialmente il nodo 0 è nel punto (x,y,x)=(200,100,0)
$node_(0) set X_ 200.0
$node_(0) set Y_ 100.0
$node_(0) set Z_ 0.0
# all'istante 2.0 secondi si muove verso il punto (300,200,0)
$ns_ at 2.0 "$node_(0) setdest 300.0 200.0 0"
```

Un'altra possibilità ancora, è quella di far muovere un nodo in modo casuale, basta abilitare il movimento random e usare la primitiva *start*:

```
$nodomobile random-motion 1
$nodomobile start
```

Non resta che generare il traffico. Come visto nel primo esempio, per creare una connessione di tipo CBR dal nodo 2 al nodo 3, che inizia all'istante 10.0, procediamo così:

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(2) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(3) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.05
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 10 "$cbr_(0) start"
```

Sempre nel pacchetto all-in-one di Ns, è contenuto uno script, *cbrgen.tcl* che, mandato in esecuzione con gli appropriati parametri, genera un file contenente la descrizione del traffico della rete. Analogamente a quanto visto per i pattern di movimento possiamo includere questo file nello script con il comando source.
Lo script completo e funzionante è contenuto nel file esempio2.tcl. Dandolo in pasto al simulatore NS otterremo due file di output. Uno con estensione .tr che contiene il tracing di ogni evento della simulazione. A questo punto basta analizzare quest'ultimo, magari con un tool esistente (ad esempio trgraph), oppure con qualche riga di codice possiamo scriverci un parser personalizzato. Ad esempio, la riga

```
r 40.647 _2_ RTR  --- 8 cbr 512 [0 0 0 0]
                        ------- [2:0 3:0 32 0] [2] 0 0
```

La *r* iniziale ci dice che si tratta di un evento di ricezione, all'istante 40.647, da parte del nodo 2, di un pacchetto CBR di 512 bytes. Un semplice programmino in Java (vedi *NSParser.java*) che calcoli la percentuale di pacchetti TCP spediti, ricevuti, e persi, e quindi ci mostri il valore percentuale del Packet Delivery Ratio, potrebbe contenere un metodo come questo:

```
try { BufferedReader fr=new BufferedReader(new
                                    FileReader(file));
  String line=fr.readLine();
  while(line!=null)
  { if(line.indexOf("tcp")>0)
     { if(line.startsWith("r")) recv++;
          else if(line.startsWith("s")) sent++;
          else if(line.startsWith("D")) drop++; }
     line=fr.readLine(); }
  fr.close();
  System.out.println("Pacchetti spediti = "+sent);
  System.out.println("Pacchetti ricevuti = "+recv);
   System.out.println("Pacchetti persi = "+drop);
   System.out.println("Packet Delivery Ratio =
            "+(((double)recv/(double)sent)*100)+"%"); }
catch(FileNotFoundException fnf)
{System.out.println(fnf.toString());}
catch(IOException ioe)
{ System.out.println(ioe.toString());}
```

Fig. 3: Dettagli su un pacchetto di acknowledgement.

Fig. 4: Le statistiche ottenute attraverso una piccola applicazione Java.

Se eseguiamo il programma passando come argomento il nome del file di trace da analizzare otteniamo il risultato voluto (vedi Fig. 4). Comunque, leggendo la parte di documentazione riguardante proprio il tracing, troverete tutti i dettagli sull'argomento. L'altro file di output, con estensione *.nam*, ci darà la possibilità, come detto, di seguire l'evoluzione della rete, il movimento dei nodi, i pacchetti spediti, (sia a livello trasporto, sia a livello di routing. Ad esempio la Fig. 2, ci mostra come, cliccando su un pacchetto, riusciamo a sapere che è di tipo TCP e di dimensione 540 bytes, e cliccando a questo punto su Monitor possiamo seguire il pacchetto per tutta la sua vita. La Fig. 3 invece ci mostra un pacchetto di *acknowledgement*. Se ci interessano invece in particolare i meccanismi di routing, possiamo seguire la loro evoluzione seguendo la propagazione delle onde da router a router.

Antonio Pelleriti

Bibliografia

• THE VINT PROJECT - "THE NS MANUAL (FORMERLY NS NOTES AND DOCUMENTATION)", *UC Berkeley, LBL, USC/ISI e Xerox PARC.*

Testare
IL CODICE CON JUNIT

Junit

Tutti sanno che testare il codice è un'operazione noiosissima, una cosa che nessun programmatore ha mai voglia di fare... E invece non è così. Non ci credete? In questa serie di articoli vi spiegheremo come scrivere codice a prova di bomba e divertirvi mentre lo fate. Tutto grazie ad una nuova, rivoluzionaria tecnica: i "test unitari".

Vi è mai capitato di passare una notte intera curvi sulla tastiera alla ricerca di un maledetto bug? Vi è mai successo che il vostro codice diventasse complesso come un piatto di spaghetti, tanto che ogni volta che correggevate un bug ne saltavano fuori altri due da qualche altra parte? Allora gioite, peccatori! Non diciamo che i tempi delle vostre sofferenze siano completamente finiti, ma se seguirete questa serie di articoli possiamo garantirvi che passerete molte più notti dormendo sereni nel vostro lettino. In questa serie di tre articoli vi introdurremo al piacere dei Test Unitari, della Grande Barra Verde e del Test-First Design. Attraverso questi strumenti potrete raggiungerete il Nirvana della programmazione, uno stato di grazia nel quale i programmatori tornano a casa dal lavoro sereni, consapevoli che il loro codice funziona e che non riserverà brutte sorprese il mattino seguente. Certo, anche noi sappiamo che i test sono una bestia antipaticissima. Tutti i programmatori sanno che dovrebbero scrivere i test, così come tutti sanno che dovrebbero perdere qualche chilo e magari smettere di fumare. Ma quante persone conoscete che testano regolarmente il proprio codice? Quasi nessuno, vero? I test sono noiosi, fanno perdere tempo, e spesso non aiutano più di tanto a trovare i bug. Almeno questo vale i test che abbiamo usato finora. Ma esiste un'altra categoria di test, diversi da quelli che conosciamo. Si chiamano "unit test" oppure, all'italiana, "test unitari". Perché sono diversi? Per cominciare sono diversi perché non testano il sistema dall'esterno, come se fosse una scatola nera, ma *da dentro*. I test unitari testano direttamente il codice, in ogni sua piccola parte. Se l'idea vi sembra stupida, aspettate di vedere come funzionano. E poi provateli.

ESEMPI DALLA VITA QUOTIDIANA

Il modo migliore per capire come funzionano i test unitari è sbirciare da sopra la spalla di qualcuno che li usa. In mancanza di meglio cercheremo di spiegarveli con qualche esempio. Questo mese potremo fare solo i primi passi. Speriamo che bastino, se non a convincerci, almeno ad incuriosirvi abbastanza da farvi leggere i prossimi due articoli di questa serie e a provare queste tecniche per conto vostro. Lo scenario: dobbiamo scrivere un programma per gestire il nostro conto corrente. Useremo la tecnica dei test unitari e la libreria JUnit (ne parliamo nelle barre laterali). La classe che vogliamo scrivere è una semplice rappresentazione di un importo di denaro in ingresso o in uscita dal conto. Dobbiamo decidere come rappresentare l'importo. Per adesso useremo due numeri per indicare rispettivamente gli Euro e i centesimi, e un valore booleano per indicare se l'importo è positivo o negativo, cioè se si tratta di una somma di denaro in ingresso o in uscita. Useremo questa rappresentazione sia all'interno della classe che nel costruttore. Ecco quindi la prima versione della nostra classe *Importo:*

```
package serie_junit;
public class Importo
{ private final boolean _positivo;
  private final long _euro;
  private final long _cent;
  public Importo(boolean positivo, long euro, long cent)
  { _positivo = positivo;
     _euro = euro + (cent / 100);
     _cent = cent % 100;
  }
}
```

Nel costruttore i centesimi oltre i 100 vengono automaticamente convertiti in Euro. I tre valori vengono poi conservati nei rispettivi campi privati. I campi sono tutti *final* (cioè costanti). Quindi l'oggetto è "immutabile", cioè non può più essere modificato dopo che è stato creato. Ora ci serve un modo per leggere il conte-

File sul CD
\soft\codice
\ExtremeProgramming\

I prossimi appuntamenti
Questa serie sarà composta da tre articoli. Nei prossimi due numeri di ioProgrammo troverete il secondo articolo ("Programmare con ritmo") e il terzo ("Abbracciare il cambiamento").

Junit

Programmazione estrema

I "test unitari" sono diventati famosi grazie alla cosiddetta Extreme Programming (o XP), una metodologia recente e molto popolare per sviluppare il software. Probabilmente in futuro parleremo ancora di Extreme Programming sulle pagine di ioProgrammo. Comunque i test unitari non sono strettamente legati all'XP, quindi potete usarli anche se l'XP non vi interessa affatto.

JUnit Uber Alles

Del framework JUnit hanno detto:
"Mai prima d'ora nella storia della programmazione così tante persone hanno avuto un debito così grande nei confronti di così poche righe di software".
Potete trovare l'ultima versione di JUnit (attualmente la 3.8.1) sul sito

http://www.junit.org

o sul CD allegato a questo numero di ioProgrammo. Per usare JUnit aggiungete semplicemente il file junit.jar al vostro classpath (oppure, se usate un IDE come JBuilder, registrate il file come libreria nel vostro progetto).

nuto di un *Importo*. Scriviamo un metodo *getValore()* che converte la rappresentazione interna in una *String*:

```
public String getValore()
{
  String risultato = "";
  if(!_positivo)
    risultato += "-";
    risultato = risultato + _euro + "." + _cent;
  return risultato;
}
```

Appena abbiamo finito di scrivere il metodo, un campanellino trilla nella nostra testa. E' la nostra coscienza, che ci ricorda che è arrivato il momento di scrivere un test unitario per verificare se il metodo *getValore()* e il costruttore funzionano. Ecco quindi la prima regola del perfetto autore di test unitari:

Ogni volta che scrivi un frammento di codice, anche se piccolo e semplice, scrivi anche un test per verificare che il codice funzioni.

Naturalmente nessun test ci garantirà mai che il nostro codice non contiene bug. Ma anche se non possiamo testare tutto, più cose testiamo meglio è. All'opera, dunque.

IL PRIMO TEST NON SI SCORDA MAI

Per scrivere il nostro primo test unitario con il framework JUnit dobbiamo scrivere una classe che eredita da *junit.framework.TestCase*. Ecco lo scheletro di un test unitario per la classe *Importo:*

```
package serie_junit;
import junit.framework.*;
public class TestImporto extends TestCase
{
  public static void main(String[] args)
  {
     junit.textui.TestRunner.run(TestImporto.class);
  }
}
```

Per lanciare la classe (che in realtà non contiene ancora nessun test) possiamo passarla al metodo statico *run()* della classe *junit.textui.TestRunner.* Per comodità abbiamo invocato questo metodo dal *main()* della stessa *TestImporto.* Proviamo a lanciare questo *main()* e vediamo cosa succede. Ecco l'output:

```
.F
Time: 0
There was 1 failure:

1) warning(junit.framework.TestSuite$1)
     junit.framework.AssertionFailedError: No tests found
                                   in serie_junit_1.TestImporto
  at serie_junit_1.TestImporto.main(Testimporto.java:14)
FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

Ci basta un solo sguardo alle ultime due righe dell'output per vedere che il nostro test è fallito. Il motivo è anche spiegato esplicitamente nel messaggio di errore: JUnit non ha trovato i test che si aspettava nella classe. Dunque dobbiamo scrivere almeno un test.
Ciascun test è un metodo *public void* il cui nome inizia con la sequenza di caratteri "test". Il TestRunner troverà e chiamerà "automagicamente" questi metodi. Dobbiamo testare il metodo *Importo.getValore()*, quindi per chiarezza chiamiamo il nostro test con il nome *testGetValore():*

```
public void testGetValore()
{
  Importo i1 = new Importo(true, 12, 06);
  assertEquals(i1.getValore(), "12.06");
  Importo i2 = new Importo(false, 0, 99);
  assertEquals(i2.getValore(), "-0.99");
}
```

Qui stiamo vedendo all'opera il succo della filosofia JUnit. Se un test restituisse lunghe sfilze di numeri, ben presto ci stancheremmo di farlo girare e di perdere del tempo per controllarne il risultato. Un test deve invece essere semplice, e deve avere un risultato booleano che possa essere verificato con un solo colpo d'occhio: o riesce, o fallisce. Per raggiungere questo obiettivo dobbiamo usare le cosiddette *asserzioni*, cioè dichiarare il risultato che ci aspettiamo di ottenere dal codice che stiamo testando. La classe *TestCase* ci mette a disposizione diverse asserzioni, e tra queste *assertEquals()*, un metodo che accetta due parametri e restituisce *true* se e solo se i due parametri sono uguali. La versione di *assertEquals()* che abbiamo usato noi accetta due parametri di tipo *long*. Ad esempio:

```
Importo i1 = new Importo(true, 12, 06);
assertEquals(i1.getValore(), "12.06");
```

In questo caso abbiamo costruito un importo e abbiamo dichiarato: mi aspetto che il metodo *getValore()* di questo importo restituisca la stringa "12.06". Possiamo usare tutte le asserzioni che vogliamo in un test. Il test riesce se e solo se tutte le asserzioni riescono, e fallisce se anche una sola asserzione fallisce. Ecco tutto questo discorso distillato in una seconda semplice regola:

I test unitari devono essere semplici. Un test unitario può avere solo due risultati: o riesce, o fallisce.

Notate che per testare il metodo *getValore()* abbiamo creato due oggetti *Importo* con parametri "quasi a casaccio", ma non del tutto. In particolare abbiamo scelto un importo positivo e uno negativo, per cercare di

non avere sorprese in nessuno di questi due casi. Torneremo sull'argomento tra poco. Intanto proviamo a lanciare il test e vediamo qual è il risultato questa volta:

```
.F
Time: 0
There was 1 failure:
1) testGetValore(serie_junit.TestImporto)
   junit.framework.ComparisonFailure: expected:<......>
                                         but was:<...0...>
  at serie_junit.TestImporto.testGetValore(
                                   TestImporto.java:22)
  at serie_junit.TestImporto.main(TestImporto.java:11)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```

Un altro errore! Questa volta non è colpa della mancanza di test. Il test c'è, ma è fallito. Cosa è successo?

NON È MAI TROPPO SEMPLICE

Purtroppo la versione di JUnit che stiamo usando dà un messaggio di errore piuttosto ambiguo quando fallisce un confronto tra stringhe, quindi non è del tutto facile scoprire dov'è il problema. Ma di sicuro sarebbe stato molto più difficile se avessimo scoperto questo bug solo tra qualche tempo, magari senza poter più sapere in quale parte del codice andarlo a cercare. Almeno in questo caso siamo sicuri che l'errore deve essere nelle poche righe di codice che abbiamo appena scritto. Riuscite a vederlo?
L'errore sta nel fatto che il nostro metodo *getValore()* non aggiunge uno zero dopo la virgola se il numero di centesimi è inferiore a 10. Quindi l'output della prima chiamata a *getValore()* non è "12.06" come ci aspettavamo, bensì "12.6".
Non è difficile correggere questo bug modificando il metodo *getValore():*

```
public String getValore()
{
  String risultato = "";
  if(!_positivo)
    risultato += "-";
  risultato = risultato + _euro + ".";
  if(_cent < 10)
    risultato += "0";
  risultato += _cent;
  return risultato;
}
```

Facciamo girare di nuovo il test:

```
.
Time: 0
OK (1 test)
```

Successo! Ci basta leggere l'ultima riga per vedere subito che il nostro metodo *getValore()* funziona. E' una vera fortuna che questo problema sia saltato fuori subito, anziché tra qualche tempo. Dite la verità: magari non ve ne eravate accorti nemmeno voi. Eppure questo codice sembrava talmente semplice che probabilmente non vi sarebbe mai venuto in mente di testarlo. Questa è la dimostrazione del fatto che anche un pezzo di codice semplicissimo può contenere dei bug – anzi, probabilmente sarete d'accordo con noi se diciamo che i bug nel codice più semplice sono anche i più difficili da trovare. Certo, scrivendo anche i test scriviamo molto più codice, quindi abbiamo più possibilità di sbagliare. Cosa succede se sbagliamo qualcosa nei test? La stessa cosa che succede se sbagliamo qualcosa nel codice: l'errore salta fuori immediatamente appena il test fallisce. Quindi è vero che potenzialmente possiamo commettere più errori, ma la gran parte di questi errori sono immediatamente visibili, e quindi innocui.

CATTIVI QUANTO BASTA

Nessun test ci darà mai la garanzia assoluta che il codice funziona al di fuori dei pochissimi casi particolari che testiamo direttamente. Diamo quindi un altro sguardo al nostro test. Siamo sicuri di non aver dimenticato niente? Abbiamo detto che un test scritto bene non deve semplicemente provare due o tre casi qualsiasi, ma deve anche cercare i casi particolarmente infelici: i casi limite, i casi particolari, eccetera. Insomma, un buon test deve essere un po' perfido. Chiediamoci ad esempio quali sono i casi limite della classe *Importo*. Quando creiamo un *Importo*, quali sono i valori massimi e minimi che possiamo passargli? Rivediamo il costruttore:

```
public Importo(boolean positivo, long euro, int cent);
```

Il primo parametro è booleano, quindi non presenta problemi. Per quanto riguarda gli altri due parametri poniamoci questi vincoli:

euro >= **0**
cent >= **0**
euro <= *Long.MAX_LONG* (massimo valore di un *long* in Java)
cent < **100**

I primi due vincoli per ora vengono lasciati al buon cuore di chi usa la classe. Se qualcuno passa dei valori negativi al costruttore di *Importo*, il codice non funziona più. Per ora supponiamo che i client della classe sappiano come usarla, quindi facciamo finta di niente (il mese prossimo risolveremo questo potenziale problema). Il terzo vincolo è sempre automaticamente rispettato, perché il compilatore impedirà a chiunque di passare come secondo parametro del nostro metodo qualcosa che sia più grande di un *long*.
Il quarto vincolo è interessante. Cosa succede se qual-

Junit

Java è, se vi pare

Tutti gli esempi di questa breve serie di articoli saranno in linguaggio Java, per tre motivi: perché la comunità Java è stata la prima a sostenere e adottare i test unitari; perché Java è un linguaggio orientato agli oggetti semplice e comprensibile; e anche perché useremo per i nostri test una libreria di nome JUnit, che è scritta in Java. Se preferite qualche altro linguaggio object-oriented, come C# o C++, seguiteci pure tranquillamente. Non dovreste comunque avere problemi a capire gli esempi, e sicuramente troverete in rete una conversione di JUnit per il vostro linguaggio preferito. Ne esistono per quasi tutti i linguaggi di programmazione terrestri, inclusi quelli meno disposti a collaborare (come Visual Basic e altri linguaggi non orientati agli oggetti). Date uno sguardo alla pagina

http://www.xprogramming.com/software.htm

Se invece ancora non conoscete alcun linguaggio object-oriented, cosa aspettate a impararne uno?

Junit

cuno crea un *Importo* passandogli un numero di centesimi maggiore o uguale a 100? Per come abbiamo scritto il codice l'operazione è legale e i centesimi in eccesso vengono automaticamente convertiti in Euro. Ma i test che abbiamo scritto fino ad ora non verificano cosa succede in questo caso particolare. Tanto per andare sul sicuro, aggiungiamo un paio di test. In uno proviamo a passare esattamente 100 centesimi (equivalenti ad 1 Euro), così ne approfittiamo anche per verificare che l'output sia corretto nel caso in cui i centesimi dell'importo sono esattamente pari a 0 (un altro caso limite). Nel secondo test scegliamo un numero di centesimi molto alto, per verificare che tutti vengano convertiti in Euro come ci aspettiamo. Notate che stiamo cercando a tutti i costi proprio i casi "sfortunati" nei quali il codice potrebbe fallire.

```
public void testGetValore()
{
 Importo i1 = new Importo(true, 12, 06);
 assertEquals(i1.getValore(), "12.06");
 Importo i2 = new Importo(false, 0, 99);
 assertEquals(i2.getValore(), "-0.99");
 Importo i3 = new Importo(true, 12, 100);
 assertEquals(i3.getValore(), "13.00");
 Importo i4 = new Importo(false, 12, 375);
 assertEquals(i4.getValore(), "-15.75");
}
```

Facciamo girare di nuovo i test.
Successo su tutta la linea! Il nostro codice si comporta secondo le aspettative. Ad esempio, 12 Euro e 375 centesimi diventano 15 Euro e 75 centesimi. Possiamo dirci soddisfatti, ed enunciare la terza regola dei test unitari:

Quando scrivi un test unitario, lascia spazio al Mister Hyde che è in te.
Cerca di trovare quei casi limite o quelle situazioni sfortunate nelle quali il tuo codice potrebbe non funzionare.

LA GRANDE BARRA VERDE

Ora che abbiamo scritto il nostro primo test unitario possiamo fare un altro salto di qualità. Finora abbiamo usato la versione testuale del TestRunner, contenuta nel package *junit.textui*. D'ora in poi useremo il TestRunner grafico del package *junit.swingui*.
Modifichiamo il metodo *TestImport.main()*:

```
public static void main(String[] args)
{
 junit.swingui.TestRunner.run(TestImporto.class);
}
```

Ora lanciamo i test. Visto che differenza?
La barra verde che vedete al centro della finestra indica che tutti i test si sono conclusi con successo. Da ora

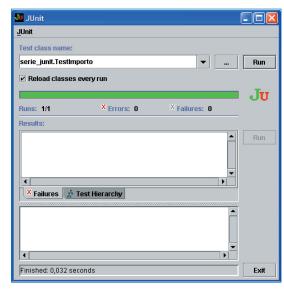



Fig. 1: La versione visuale del TestRunner.

in avanti, questa barra verde sarà il nostro passaporto per il paradiso. Un programmatore abituato a scrivere i test unitari per il proprio codice può tenere il TestRunner aperto sullo sfondo e far girare i test in continuazione, anche ogni volta che compila. Se appare la barra verde, si va avanti. Se invece un test fallisce la barra diventa rossa. Allora ci si ferma e si corregge il codice (o il test) malfunzionante. Quando vi sarete abituati a questo modo di lavorare non accetterete compromessi: solo se il vostro codice è completamente testato e se vedete una barra verde andrete avanti tranquilli. La pace e la serenità che i test unitari vi daranno sarà una delle sensazioni più piacevoli che potrete sperimentare nella vostra attività di programmatori.
Ora sappiamo anche perché i test devono essere booleani. Solo così potranno essere verificati da un sistema automatico, e daranno un risultato visibile al primo sguardo e non ambiguo. Di conseguenza saremo invogliati a far girare tutti i test ogni volta che facciamo una modifica al codice, per quanto piccola. Quello che era un compito gravoso diventerà un piacere, perché basterà premere un pulsante per avere una "pacca sulla spalla virtuale" nella forma di una barra completamente verde, che ci rassicurerà circa il corretto funzionamento del nostro sistema.
E così siamo arrivati alla quarta e ultima regola del perfetto autore di test unitari:

Lancia i test in continuazione.
Se vedi una barra verde, vai avanti tranquillo.
Se vedi una barra rossa, fermati e correggi il problema dovunque esso sia – nel codice o nei test.

Ci state prendendo gusto? Allora arrivederci al prossimo numero di ioProgrammo, dove affronteremo un argomento che di solito non viene associato alla programmazione dei computer: il ritmo.

Paolo Perrotta

Troppe lingue

Nel codice dei nostri esempi usiamo nomi per metà in inglese e per metà in italiano, come *getValore()*. Mescolare le due lingue è di solito una cattiva idea, e alla lunga può rendere il codice piuttosto confuso. Ma in questo caso vogliamo rispettare le convenzioni di Java, come quelle sui metodi "get" e "set", e allo stesso tempo vogliamo facilitare la vita ai lettori che non hanno molta pratica con l'inglese. Quindi abbiate pazienza, e rassegnatevi a vedere altri nomi in "lingua mista" nel corso di questa serie. Se conoscete bene l'inglese, vi consigliamo di non usare l'italiano nel vostro codice se non nei commenti.

Il Software
sul CD-ROM di ioProgrammo

Ecco i migliori software scelti per voi da ioProgrammo: Borland JBuilder 8 trial, XML Spy 5.0 Professional Edition, AppMentor 3.0 for Visual Basic.

Borland JBuilder 8 trial

Borland ha da poco rilasciato la nuova versione di JBuilder 8.0. Questa nuova release è caratterizzata da una struttura completamente rinnovata e basata sul framework open source Jakarta Struts. Da parte degli sviluppatori è stata data una notevole importanza alle funzioni di collaborazione, testing e di debugging, ora, per esempio, l'Ide può eseguire lo "spot debug", solo su una piccola parte del codice (per poi modificarlo nell'immediato), su JSP o anche su codice non Java. Le funzioni di collaborazione, per lo sviluppo in team aziendali, invece, si avvalgono dei collaudati tool di Rational: Clear Case, per la gestione e l'integrazione del codice, CVS per la gestione delle versioni e del codice sorgente. In aggiunta JBuilder 8 supporta tutte le ultime versioni degli application server di Borland, Bea Systems, Oracle e Sun Microsystems. Sono inoltre supportati i sistemi Aix di Ibm e la piattaforma Hp-Ux. Per l'attivazione del prodotto si richiede la registrazione sul sito Borland (www.borland.com).
Borland_JB

AUTOFORM@asp

Sviluppato da una software house italiana, AUTOFORM@asp è un interessante tool di sviluppo RAD che, poggiandosi su tecnologie consolidate e molto diffuse, permette di rendere più rapide molte fasi della progettazione di applicazioni gestionali, oltre a garantire la coerenza estetica delle interfacce prodotte. Autoform consente la creazione di applicazioni per Internet, reti locali e stand alone garantendo la semplificazione dell'interfacciamento con Microsoft SQL Server e Oracle, la generazione di applicazioni multilingua, l'interfacciamento degli OLAP Services di Microsoft, la gestione sicura delle transazioni, numerose funzioni per l'export dei dati. L'ambiente autoform è composto da una serie di strumenti che si possono racchiudere in due grandi categorie: strumenti lato server e strumenti lato client. Dalla parte server abbiamo un motore di database, un'applicazione che fa da interfaccia tra il database e l'applicazione client e un generatore di report. Lato client troviamo l'applicativo che fa da interfaccia utente e che comunica con il componente server tramite rete locale o via Internet. È possibile installare Autoform in tre modalità differenti:

- *Server*: è la tipologia di installazione naturale, richiede Microsoft NT o 2000, e permette la connessione contemporanea di più client in una LAN;
- *ASP*: permette l'accesso al server via Internet utilizzando il protocollo http;
- *Standalone*: installazione e utilizzo su una singola workstation.

Versione dimostrativa.
Autoform

Ariacom Business Reports 1.6

Un potente ambiente per la creazione di report automatizzata. Attraverso la creazione dinamica di query SQL, anche gli utenti meno esperti potranno gestire e trarre informazioni dai più complessi database relazionali. Ariacom Business Reports può coprire vasto spettro di esigenze: dalla pubblicazione di una directory su Internet , all'analisi multidimensionale di un data-warehouse. Le funzioni principali sono: generazione e ottimizzazione dinamica delle query, avanzate funzioni di report, limitazione all'accesso dei dati attraverso un'attenta gestione della sicurezza, possibilità di generare report in diversi formati (html, csv, txt) e su dispositivi diversi (file, stampante, fax, email) e molto altro.
bfree16.exe

Ulead Cool 3D Studio 1.0

Che dobbiate realizzare il nuovo logo della vostra software-house, un filmato introduttivo sul vostro nuovo prodotto, o semplicemente una gif animata che non sia banale, Ulead Cool 3D Studio è il prodotto che fa per voi! La creazione di grafica e testo animata raggiunge in questi prodotto livelli di semplcità ed eccellenza mai visti in precedenza e, grazie alla grande compatibilità garantita in output, sarà possibile distribuire i filmati che produrremo in molteplici formati: sequenza di fermo immagini, animazione GIF e filmati Flash. Si integra perfettamente con gli altri prodotti Ulead per l'editing Video come MediaStudio Pro e VideoStudio. Versione di valutazione valida trenta giorni.
Ulead_Cool3D

Code Counter Pro 1.0

Una comoda applicazione che consente a noi programmatori di quantificare il lavoro che abbiamo svolto o stiamo svolgendo. Code Counter si occupa di valutare il numero di righe, spazi bianchi, commenti e quant'altro sia interessante per effettuare statistiche sul codice. I linguaggi supportati sono numerosi: C/C++, Java, Delphi /Pascal, VB, PHP, ASP ed altri ancora, è possiblile effettuare analisi anche su porzioni di testo che non siano riconducibili a codice sorgente. Versione di prova valida trenta giorni, presenta alcune limitazioni nel suo utilizzo.
ccountp.exe

Installazione ActiveX in Delphi

Dal menu "Component" selezionare la voce "Import ActiveX Component"; nella schermata presente a video è visibile una list box contenente l'elenco degli ActiveX installati nel sistema; da questi è possibile scegliere il componente e installarlo mediante il bottone Install. Tramite l'uso dei bottoni "Remove" e "Add" è possibile rispettivamente rimuovere un componente dalla list box o aggiungerne uno non presente in essa ma comunque residente in qualche directory del sistema. Dalla list box "Palette Page" si seleziona la sezione (Standard, Additional, Win32, etc, etc) nella quale troverà posto l'icona del componente.

Code Magic 4.2

"Ma non l'avevo già risolto?" questa la tipica domanda di noi sviluppatori quando ci troviamo di fronte ad un problema che ci sembra di aver già affrontato, ma non ricordiamo quando, ne' come. Questa applicazione ci viene in aiuto proprio in situazioni come queste. Grazie ad un parser particolarmente efficace, riesce ad archiviare tutto il codice che produciamo all'interno di una vista ad albero che permette un facile reperimento in fase di produzione del codice. Il syntax highlighting supporta i più noti linguaggi di programmazione.
cm_setup.exe

Dev-Pascal 1.9.2

Un completo e gratuito ambiente di sviluppo per Pascal, dotato di tutte le più utili caratteristiche degli IDE moderni. Tra le funzioni disponibili segnaliamo: colorazione del codice personalizzabile, wizard per la creazione di pacchetti di installazione, supporto per la creazione di DLL, sviluppo assistito per le interfacce, possibilità di impostare dei template per la produzione rapida di codice. Nella directory è anche presente un debugger che si integra perfettamente nell'ambiente di sviluppo.
DevPascal

Expert Troubleshooter 5.0

Basato su un motore di intelligenza artificiale, Export Troubleshouter consente di costruire applicazioni di monitoraggio per scoprire immediatamente gli eventuali malfunzionamenti di apparecchi elettronici ed elettromeccanici. Sono disponibili diversi approcci: dalla verifica manuale assistita dal PC, alla generazione di pattern per l'individuazione degli ambiti critici. Particolarmente interessante risulta essere la generazione automatica di codice in Basic, C e Pascal per la creazione di applicazioni di test.
ETSetup

FontLab 4.52

Un sofisticato editor di font che consente di importare, esportare, creare e modificare font nei loro formati nativi. Inoltre, grazie ad un tool di disegno vettoriale, è possibile intervenire con precisione nella modifica e nella creazione di font personalizzati. Questa nuova versione include una potente sistema di macro. Versione dimostrativa.
FL4WinDemo.exe

Hephaestus 2.01

Un completo kit per la realizzazione di giochi di ruolo che permette di scrivere la propria avventura attraverso un semplice linguaggio di scripting e di condividere il gioco con chiunque altro abbia installato Hephaestus. Essendo scritto interamente in Java, Hephaestus gira su qualsiasi piattaforma. Gratuito.90
hephaestus.zip

IBasic Standard Version 1.99c

Una occasione per chi si avvicina al mondo della programmazione: Ibasic consente di scrivere applicazioni stand alone per Windows con estrema semplicità, adottando una sintassi sovrapponibile in buona parte al Basic standard. Il linguag gio è sufficientemente completo, comprendendo oltre 180 fra comandi e funzioni. In questa nuova release, è stata migliorata del 150 percento la velocità di esecuzio ne, sono state inserite nuove funzioni per la gestione dei file ed è stata migliorata l'interfaccia utente.
ibasic.zip

Instant Report 1.4

Un generatore di query SQL che consente la costruzione completamente visuale dei report. Concepito per essere il più semplice possibile, permette di realizzare report anche complessi con pochissimi colpi di mouse. Compatibile con tutti i più diffusi database sia free che commerciali. Gratuito.
ireport-1.4.zip

NexJDE 1.0a

Una comoda applicazione che va ad collocarsi nella categoria degli Universal Data Access tool: consente di effettuare query e ottenere risultati a partire da dati immagazzinati in un qualsiasi database: Oracle JDBC, SUN JDBC ODBC, PostgreSQL, mySQL, Sybase, e MS SQL Server. I risultati possono essere mostrati in una tabella HTML o all'interno di un'apposita interfaccia utente.
JDELight.zip

Metamill 2.2

Un software per la modellazione software UML completamente visuale. Metamill risulta particolarmente semplice da usare, in special modo se paragonato ad altri software di modellazione UML. Tra le caratteristiche più interessanti c'è il repository, attraverso cui più sviluppatori possono collaborare utilizzando i medesimi elementi senza "pestarsi i piedi". Una volta sviluppato il modello, Metamill è in grado di produrre il codice relativo sia in C++ sia in Java. Shareware.
Mmill221.exe

MyTool 1.2

Un interessante front end per MySQL comprensivo un editor per effettuare tutte le più comuni operazioni di gestione di DBMS per via visuale. Interessante la funzione di monitoraggio dello stato delle query sul server. Versione dimostrativa.
mytool-setup.exe

Pro-Test 1.0

Una interessante applicazione che consente di velocizzare e rendere più efficiente una delle parti più noiose della creazione del software: il testing. Pro-Test si occupa di generare casi di test per analizzare la robustezza del codice che scriviamo. Usato con attenzione, Pro-Test può migliorare e velocizzare sensibilmente la produzione di codice. Versione di prova valida dieci giorni.
Pro-Test_10Day_Trial.EXE

ResRipper 3.8

Davvero comodo questo tool che consente di analizzare velocemente le risorse immagazzinate nei

Risorse Java

Molte delle risorse Java riportate all'interno del CD ROM sono munite di file .java, .class e di file html per essere testate. Nel caso di compilazione del file .java si dovrà utilizzare un opportuno strumento, come ad esempio il JDK di Sun. Per utilizzarlo si dovrà operare da prompt del DOS, accedere alla directory bin dell'ambiente stesso ed avviare il Java Compiler digitando la stringa: javac "nomefile".

Installazione componenti VC++/C++

I file C++, delle librerie facenti parte del CD-ROM, possono essere direttamente inclusi all'interno dei vostri progetti. Le risorse Visual C++ sono invece fornite del file progetto, quindi dall'ambiente di sviluppo Microsoft utilizzare la voce di menu "Open Project" ed automaticamente tutti i file afferenti al progetto stesso verranno visualizzati, e da tale ambiente è possibile inoltre eseguire l'applicativo stesso. Se invece risorsa riportata è una dll, allora essa potrà essere direttamente inclusa all'interno di un progetto.

Installazione ActiveX in Visual Basic

Dal menu Progetto selezionare la voce Componenti (CTRL+T); nella schermata presente a video è visibile una list box contenente l'elenco dei componenti ActiveX installati nel sistema; da questi è possibile selezionare uno o più componenti e confermare mediante il bottone OK; qualora il componente non fosse installato nel sistema ma fosse comunque presente nel computer è possibile selezionare quest'ultimo tramite l'utilizzo del bottone "Sfoglia" mediante il quale si ha accesso alle directory del sistema; da queste è possibile localizzare il componente da installare.

nostri dischi e che sono normalmente nascoste: migliaia di immagini (BMP, DIB, GIF, JPG) icone, filmati (AVI, MPG, GIF animate), font e suoni. Tutte risorse normalmente bloccate e nascoste all'interno delle applicazioni e che, grazie a ResRipper, possono ora essere individuate, estratte e riutilizzate nei nostri progetti.
Versione di prova valida trenta giorni.
resripper_trial_version.exe

Stylus Studio 4.5

Forte dei molti premi vinti e di una base di installato che conta oltre 30.000 sviluppatori sparsi in tutto il mondo, Stylus Studio si presenta come un potente tool visuale per lo sviluppo e la gestione di documenti XML e XSL. Rimarchevole la qualità della mappatura dei documenti che Stylus Studio offre per via grafica, consentendo una più facile analisi dei documenti. Permette di gestire e modificare tutti i tipi di file che appartengono ad un'applicazione XML: oltre a documenti XML, abbiamo il pieno supporto per gli XSLT stylesheets, DTD e gli XML schema, oltre ai file Java. Gira su Windows NT - 2000 -XP. Tra le funzioni più interessanti, segnaliamo la possibilità di effettuare il debug e la presenza di una potente funzione di preview sulle trasformazioni realizzate. Un prodotto che può risultare valido sia ai principianti che agli esperti e si presta anzi a fare da guida in un percorso di apprendimento delle tecnologie legate a XML. All'atto dell'installazione è richiesto un collegamento Internet al fine di riempire una breve form. Una chiave di attivazione verrà inviata alla propria casella di posta elettronica.
StylusStudio

UltraEdit-32 9.2

UltraEdit-32 è principalmente un editor esadecimale con un completo supporto per le macro e numerose funzioni avanzate come la conversione di file da DOS a Unix. In questa nuova versione sono state aggiunte delle comode funzionalità di autocompletamento, oltre ad un miglioramento complessivo dell'interfaccia.
Versione di valutazione valida 45 giorni.
uedit32.zip

XMLEditPro 2.0

Semplice ed intuitivo, questo editor XML ha giusto l'essenziale per la creazione e l'editing di documenti XML. Proprio questo suo essere spartano, lungi dall'essere un limite, diventa un pregio se visto sotto l'ottica della semplicità dell'interfaccia e della velocita. Gratuito.
xep2setup.exe

XML Spy 5.0 Professional Edition

Il più avanzato strumento di sviluppo per XML. Uno dei punti di forza risiede nella possibilità di switchare velocemente fra quattro differenti viste per ogni documento: *Enhanced Grid* per il dettaglio di tutti gli elementi: *Schema Design; Text View* per una programmazione a più basso livello e una vista di preview che anticipa come il documento sarà visualizzato da un comune browser. E' possibile sviluppare sia documenti XML che DTD che saranno poi pubblicabili via FTP attraverso lo stesso XML Spy.
Davvero eccezionale il supporto offerto ai Web Services grazie ad apposite interfacce per i protocolli SOAP, XSL e WSDL.
XmlSpy

Dev-C++ 5 Beta 7 (4.9.7.0)

Uno dei migliori ambienti gratuti di sviluppo per C++. In questa versione, l'interfaccia raggiunge livelli di eccellenza: scrivere, compilare, eseguire ed effettuare il debug, tutto molto efficiente e con la possibilità di avere delle icone stile Gnome. Quasi una scelta obbligata per chi vuole sviluppare applicazioni per win32 e non vuole affrontare la spesa di un Visual C++. Tra le caratteristiche più interessanti: completamento automatico del codice, manager di progetto, syntax highlighting personalizzabile, creazione assistita di librerie statiche e DLL, supporto per i packages e molto altro ancora. Provatelo.
devcpp4970-gcc32.exe

Visual Paradigm for UML Community Edition

Un ambiente di sviluppo integrato per UML che farà la gioia di tutte le figure coinvolte nello sviluppo di applicazioni di grandi dimensioni. Completamente gratuito, non rinuncia ad una ottima interfaccia di livello professionale e copre tutte le maggiori esigenze di chi si occupa di UML.
Da provare.
VParamUML

AppMentor 3.0 for Visual Basic

Una piattaforma che consente di velocizzare la produzione di nuove applicazioni Visual Basic, grazie ad un approccio template-oriented che consente di ridurre i tempi di sviluppo anche dell'80 percento. L'accesso ai database è realizzato attraverso una logica n-tier che consente di demandare buona parte della gestione a basso livello. Che siate sviluppatori esperti o "apprendisti stregoni", AppMentor può essere la soluzione giusta per fare un passo decisivo verso lo sviluppo di applicazioni professionali.
Versione di valutazione.
AppMentor3

ActivePerl 5.8

La versione Windows di una completa distribuzione Perl. Otre al linguaggio, potrete sperimentare il Perl Package Manager (PPM) che, attraverso una semplice interfaccia e riga di comando, permette di installare e gestire moduli ed estensioni. La copiosa documentazione di cui è fornita questa distribuzione varrebbe da sola a installare il pacchetto.
Gratuita.

ActivePerl-5.8.0.804-MSWin32-x86.zip

LE FAQ DI IOPROGRAMMO

Le risposte alle domande più frequenti

Ogni mese troverete riportate le domande che più spesso giungono in redazione. Capita frequentemente che, affrontando linguaggi in continua evoluzione, si diano per scontati alcuni concetti o alcune caratteristiche di base: queste pagine sono l'occasione per ribadire o spiegare meglio queste nozioni.

Java

Cosa accomuna Java e C#

Quando fu presentato, il C# sollevò molte perplessità per l'eccessiva somiglianza con Java. Addirittura si ventilò un'accusa di plagio, poi caduta nel vuoto, soprattutto perché, come spesso accade in questi casi, molte delle caratteristiche che C# avrebbe "mutuato" da Java erano in realtà preesistenti a Java stesso, della serie: "chi non ha peccato scagli la prima pietra"!
Ormai le polemiche sono spente e C# si è dimostrato semplicemente un nuovo (e ottimo per la verità) linguaggio, e ci sembra interessante, a mente fredda, riassumere brevemente i principali punti di contatto fra il gioiello di Sun e l'astro nascente di Microsoft. Eccone un sommario elenco:

- Entrambi prevedono la compilazione in un codice indipendente dalla piattaforma, codice che viene eseguito da una macchina virtuale.
- Un meccanismo di Garbage Collection e la scomparsa dei puntatori (fondamentali in C++, il C# ne permette l'uso in particolari porzioni di codice dichiarate esplicitamente unsafe).
- Potenti meccanismi per la reflection.
- Assenza di file Header (anch'essi fondamentali in C++).
- Tutte le classi derivano dalla classe base Object.
- Supporto per i thread attraverso un meccanismo di lock attivato sulle porzioni di codice marcate come locked /synchronized.
- Supporto per le interfacce con ereditarietà multipla per le interfacce stesse e singola per le implementazioni.
- Disponibilità delle inner class.
- Assenza di funzioni o costanti globali: tutto appartiene al dominio delle classi.
- Uso della "notazione punto" al posto degli operatori "->" e "::" utilizzati in C++.
- Tutti i valori sono inizializzati prima di poter essere usati.
- Impossibilità si usare gli interi per valutare le condizioni di *if*.
- I blocchi di Try/Catch possono avere la clausola *finally*.

Cosa sono gli obfuscator?

Il bytecode racchiude le informazioni che descriviamo con il codice sorgente Java. Il bytecode è generato dal compilatore Java ed è relativamente semplice risalire al codice sorgente attraverso un'operazione di reverse engineering: molti decompilatori, alcuni pubblicati anche nei CD allegati a ioProgrammo, consentono infatti di passare dal bytecode al codice sorgente in maniera pressoché automatica, e con un buon grado di approssimazione. La proprietà intellettuale legata al codice da noi prodotto è dunque resa abbastanza vulnerabile. Gli obfuscator (offuscatori) rendono il lavoro dei decompilatori più arduo, ed il codice che si riesce a tirar fuori è meno leggibile e meno utile. Esistono essenzialmente due tipi di obfuscator: una prima categoria si occupa di cambiare i nomi di classi, campi e metodi con sequenze di caratteri senza senso, con l'obiettivo di disorientare e far perdere il senso complessivo del progetto. Ciò che rimane in chiaro è la struttura dell'applicazione che, con un più attento lavoro di reverse engineering, può comunque essere "riutilizzata". Una seconda categoria di obfuscator, oltre ad offuscare i nomi, si occupa di offuscare anche il flusso di esecuzione dell'applicazione, attraverso delle leggere modifiche al bytecode che rendono oscuro il flusso di esecuzione dell'applicazione senza (ovviamente) modificarne il comportamento. Tipicamente gli offuscatori di questo tipo intervengono nelle strutture di selezione e nei costrutti di loop, modificandoli in maniera tale da non consentire la ricostruzione del sorgente Java equivalente.

Perché dichiarare un metodo "synchronized"?

La sincronizzazione dei metodi in Java consente la realizzazione di sezioni critiche nella programmazioni multithread. Sezioni cioè in cui è necessario garantire che la concorrenza fra i vari thread sia in qualche modo "sospesa". Un problema che si presenta tipicamente nell'accesso a risorse condivise; mettiamo il caso che due applicazioni vogliano scrivere sull'hard disk: per garantire la consistenza dell'informazione, è ovviamente necessario che quale delle due cominci per prima a salvare i dati non venga interrotta dall'altra. In questi casi si adotta una sorta di "lucchetto" virtuale associato alla risorsa (in questo caso l'hard disk): se l'applicazione App1 vuole scrivere sul disco, deve innanzitutto procurarsi il lucchetto dell'hard disk. L'applicazione App2, che pure vuole scrivere, tenta di procurarsi il lucchetto dell'hard disk ma deve attendere, poiché il lucchetto è detenuto da App1. Quando App1 avrà finito le sue operazioni di scrittura, avrà cura di rilasciare il lucchetto. Lucchetto che verrà intercettato da App2, che inizierà le sue operazioni di scrittura, e così via. Ecco spiegata, in termini poco ortodossi, l'utilità della nozione di "lucchetto" nella programmazione, multithread. Un metodo è dichiarato synchronized secondo la sintassi:

```
public synchronized int scriviValore()
{  ...
}
```

Grazie a questa dichiarazione, ad ogni istanza della classe cui il metodo appartiene sarà associato un lucchetto. Nel momento in cui un thread invocherà il metodo *scriviValore()*, esso diverrà detentore del lucchetto associato alla classe di cui *scriviValore()* è metodo. Il possesso di questo lucchetto durerà fino alla completa esecuzione del metodo *scriviValore()*, ciò significa che, per tutto questo tempo, nessun altro thread potrà invocare un metodo synchronized di quello stesso oggetto. Se il metodo, oltre ad essere dichiarato synchronized, è definito anche come static, il lucchetto sarà esteso all'intera classe e non più alle singole istanze.

Perché dichiarare una classe abstract?

La parola chiave abstract serve a indicare una classe o un metodo che non ha implementazione. Quando una classe è dichiarata *abstract*, essa non può essere istanziata. Solo le sottoclassi di una classe astratta possono essere istanziate, sempre che non siano state dichiarate *abstract* a loro volta. Similmente, dichiarando un metodo come *abstract*, si specifica che il metodo non è implementato nella classe in cui è dichiarato, ha solo la signature:

```
abstract class Risorsa {
  // ...
  public abstract int leggiValore();
}
```

Un metodo astratto esiste solo allo scopo di essere sovrascritto (overridden) nelle sottoclassi della classe in cui è dichiarato. E' bene tenere presente che un metodo astratto può essere dichiarato solo nell'ambito di una classe astratta, mentre non è obbligatorio che una classe astratta presenti dei metodi astratti. Nel momento in cui si dichiara una sottoclasse di una classe astratta, la sottoclasse deve necessariamente fornire una implementazione dei metodi astratti della superclasse, o in alternativa essere dichiarata a sua volta come classe astratta.

Qual è il ruolo di static in Java?

Quando dichiariamo static una variabile di una classe, quella varibile viene istanziata una sola volta, qualsiasi sia il numero di istanze create a partire da quella classe.
In pratica, se un'istanza della classe cambia il valore di quella variabile statica, il cambiamento si riflette in tutte le altre istanze della classe. Se invece dichiariamo un metodo static, il campo di azione del metodo si sposta dalle istanze della classe alla classe stessa. Un metodo statico può fare riferimento solo a variabili e metodi statici e non accetta l'override nelle classi derivate dalla classe di appartenenza: un metodo static è implicitamente definito final.

Cos'è un file JAR ?

JAR è un acronimo e sta per Java ARchive. Un file .jar è semplicemente un archivio di file compressi. In un file JAR è possibile sistemare tutte le classi necessarie ad un'applicazione con il vantaggio di una notevole riduzione dello spazio occupato dalla versione del software che distribuiremo. Supponiamo di voler creare un file .jar, chiamato archivio.jar, in cui sistemeremo tutti i file presenti in una directory.
Ecco la sintassi:

```
jar -cf archvio.jar *.*
```

Lo switch –c è servito a indicare la volontà di creare un nuovo archivio, mentre con –f abbiamo la possibilità di specificare il nome del file. Per avere il dettaglio delle operazioni compiute da Jar, possiamo aggiungere lo switch –v.

```
jar -cvf ajarfile.jar *.*
```

In questo modo appiamo attivato l'opzione "verbose", con il risultato di avere in output tutte le informazioni riguardo ai file inseriti nell'archivio. Per estrarre il contenuto di un file Jar, la sintassi è la seguente:

```
jar -xf ajarfile.jar
```

È possibile visualizzare l'elenco completo delle opzioni disponibili avviando il comando jar, senza fornire alcun parametro.

Cos'è la Extreme Programming?

Con Extreme Programming (XP) si intende una serie di regole e principi per lo sviluppo rapido di software professionale. Lo scopo è dunque duplice: massima velocità nella realizzazione e massima qualità possibile del prodotto finito.
XP rappresenta una sorta di metodologia, anche se una delle più leggere e semplici da seguire, che può essere ripetuta ogni qual volta si debba sviluppare un'applicazione. Vediamo brevemente le dodici principali regole della Exreme Programming:

1) **Il punto di partenza è rappresentato dalla scrittura delle cosiddette "User Stories".** In questa fase i committenti e gli sviluppatori collaborano per la definizione specifiche funzionali e del tempo necessario alla realizzazione dell'applicativo. I committenti sono chiamati a definire scrivere queste storie: ognuna rappresenta una funzionalità del sistema. Le singole storie saranno esaminate dagli sviluppatori al fine di valutare il tempo di sviluppo necessario. Normalmente, se una storia impiega più di tre settimane, si considera troppo grande e va spezzata in più sotto-storie, se invece richiede meno di una settimana, vuol dire che si è scesi ad un livello di dettaglio eccessivo e la storia va accorpata con un'altra.

2) **Rilasciare nuove versioni il più presto possibile.** Rilasciare spesso al cliente nuove versioni, aggiungendo anche poche caratteristiche per volta, consente di avere un continuo feedback sul lavoro e permette di correggere immediatamente eventuali errori della progettazione.

3) **Scegliere una metafora per il sistema che si va implementando.** La metafora consente di ricordare facilmente i nomi degli oggetti coinvolti nel progetto o consente di "indovinarli" se non li si ricorda.

4) **Usare sempre il design più semplice possibile.** Le richieste possono cambiare da un giorni all'altro e dunque è bene progettare con l'obiettivo di soddisfare solo le richieste attualmente formulate. Inoltre, il fatto che l'XP sia un processo iterativo, consente di proseguire per affinamenti successivi.

5) **Testare continuamente.** Prima di scrivere una nuova funzione, il programmatore scrive un test per metterla alla prova. Il lavoro si considera fatto quando i test hanno esito positivo. Esistono due tipi di test:

a. **Unit Test:** sono test automatici scritti dagli sviluppatori per provare il codice mentre lo scrivono. Generalmente ogni Unit Test prova una singola classe o un piccolo insieme di classi, inoltre per effettuare gli Unit Test si adottano tipicamente dei framework specifici come il JUnit (descritto nell'articolo dell'ottimo Paolo Perrotta presente in questo stesso numero di ioProgrammo)

b. **Acceptance Test:** noti anche come test funzionali, sono definiti dai committenti allo scopo di verificare che il sistema rispetti complessivamente le specifiche. I test funzionali si occupano in genere di verificare l'intero sistema o larghe porzioni di esso. Un tipico test consiste in uno script equivalente alle azioni di un comune utente di fronte al sistema e nella verifica che il risultato sia coerente con le aspettative.

6) **Cancellare sempre il codice superfluo.** Spesso si è portati a mantenere delle vecchie funzioni o classi non più necessarie, o anche pezzi di codice che ormai non utilizziamo più. L'XP consiglia di eliminare tutto il superfluo il più spesso possibile. Grazie ai continui test abbiamo la garanzia di non buttare via niente di utile e possiamo dunque procedere ad un refactoring continuo.

7) **Programmazione in coppia.** Tutto il codice è prodotto da due programmatori seduti alla stessa workstation. In pratica, questo "trucco" consente di avere il codice revisionato nel momento stesso in cui viene prodotto. Può apparire una misura eccessiva ma è stato dimostrato che il codice prodotto da due programmatori che lavorino assieme è migliore e più consistente del codice prodotto dagli stessi separatamente. Tipicamente, il programmatore che scrive "fisicamente" il codice bada alla correttezza sintattica, mentre l'altro cerca di tenere uno sguardo di più ampio respiro, controllando la correttezza semantica di quanto si va sviluppando.

8) **Il codice è di tutti.** Nessuna persona e nessuna coppia può dirsi proprietaria di un pezzo di codice. E' riconosciuto a tutto il team il diritto di agire su qualsiasi parte del codice. Chiunque può aggiungere funzionalità ed effettuare debugging su qualsiasi pezzo di codice in qualsiasi momento. Questa che potrebbe apparire una libertà eccessiva è possibile e assolutamente sicura grazie alla presenza dei test: chiunque apporti una modifica, deve garantire che essa superi i test previsti per quella porzione di codice.

9) **Integrare continuamente.** Gli sviluppatori devono integrare le modifiche che apportano al più presto possibile e, in ogni caso, non deve passare più di un giorno prima che le modifiche vengano integrate nel sistema. Solo così è garantita la consistenza fra tutte le unità di codice, oltre a permettere a tutti gli sviluppatori di lavorare sempre con la versione più aggiornata del codice.

10) **Si lavora non più di quaranta ore a settimana.** Tutti i programmatori vanno via al momento giusto, solo in situazioni critiche è contemplata una settimana di straordinari. Se cominciano a presentarsi più settimane consecutive di straordinari, vuol dire che c'è qualcosa da rivedere nel processo di produzione.

11) **Il cliente deve essere sempre disponibile.** È necessario che ci sia un continuo scambio di idee fra gli sviluppatori e il committente che non deve dunque limitarsi a presentare le User Stories all'inizio del progetto, ma deve rappresentare un punto di riferimento costante per tutto il team. Le User Stories sono infatti dei canovacci molto approssimativi ed è necessario un continuo feedback del cliente per indirizzare correttamente lo sviluppo del progetto.

12) **Codice standard.** Il codice deve essere scritto rispettando standard comuni accettati da tutti. In pratica, leggendo diverse porzioni di codice, non dovrebbe essere possibile distinguere la "mano" del programmatore. Questa coerenza formale consente una facile leggibilità ed un semplice accesso a tutti gli sviluppatori verso qualsiasi porzione del progetto.

Quando usare Private per le dichiarazioni di variabili e di procedure?

Ogni volta che ciò è possibile, è bene usare Private per le dichiarazioni. Dichiarando Private le procedure e le variabili di un modulo, infatti, queste saranno accessibili soltanto nel modulo stesso:

```
Private variabile As Integer
Private Sub Procedura()
```

Conoscere a priori se una variabile o una procedura debba essere dichiarata Private o Public non è facile, specialmente se non si è valutato bene il problema al momento della redazione del codice. Se la variabile o la procedura è dichiarata Public quando non dovrebbe esserlo, si potrebbero creare problemi di eccessi di visibilità dell'oggetto in questione. Per fortuna, molti di questi problemi possono essere risolti con un'analisi automatica del codice.

Come si può ottimizzare la velocità di visualizzazione?

La velocità di visualizzazione di un programma è spesso più importante della sua reale velocità di esecuzione. Ecco alcuni suggerimenti per aumentare la velocità di visualizzazione di un'applicazione:

- Impostare la proprietà *ClipControls* sul valore *False*.
- Usare il controllo *Image* invece di *PictureBox* e *Label* invece di *TextBox* quando possibile.
- Nascondere i controlli quando si impostano le proprietà; ciò impedisce di effettuare ridisegni multipli.
- Mantenere nascosto un form precedentemente caricato per visualizzarlo velocemente. Questo metodo richiede un dispendio cospicuo di risorse di memoria.
- Usare priorità basse per i processi lun-

ghi. Si possono realizzare processi a bassa priorità di esecuzione collocando opportunamente dei controlli *DoEvents*. Un ottimo posto dove collocare i *DoEvents* sono i cicli interni. Bisogna tenere presente, però, che l'utente potrebbe premere qualche tasto. Il processo a bassa priorità deve tener conto di ciò che l'utente può o non può fare durante la sua esecuzione. Impostare quindi le proprietà *Processing* col valore *True*.

- Usare barre di avanzamento di stato.
- Pre-caricare dati che serviranno successivamente. Per esempio, potete caricare i contenuti di una tabella di un database in un vettore.
- Usare la caratteristica *Show* degli eventi *Form_Load*.
- Semplificare lo startup form o usare uno splash screen.

C++

Quando vengono invocati i distruttori per le variabili locali?

Le variabili *"non-static"* vengono distrutte automaticamente quando l'esecuzione esce dal loro ambito di visibilità. Senza voler approfondire troppo il concetto di visibilità, possiamo affermare che l'ambito di vita di una variabile termina subito dopo il simbolo *"}"* in cui la variabile è stata dichiarata. Al momento dell'uscita da tale ambito, le variabili vengono distrutte dal compilatore chiamando il distruttore appropriato nella loro classe di appartenenza. Se l'oggetto ha allocato memoria (ad esempio un array), la distruzione rilascia la memoria precedentemente impegnata:

```
class vettore
{
private:
float *ptr;
public:
// costruttore
vettore(int n) { ptr = new float[n]; }
// distruttore
~vettore() { delete [] ptr; }
};
main()
{
// ...
{vettore v(5); // alloca la memoria
// operazioni...
} // fine dell'ambito di visibilità
                                dell'oggetto vettore
// l'oggetto vettore viene qui distrutto,
                         e la memoria viene rilasciata
// ...
}
```

Come funzionano le espressioni separate da virgola?

Le espressioni separate da virgola derivano dal linguaggio C. Spesso queste istruzioni sono impiegate all'interno di clicli *for* e *while*. Le regole sintattiche di queste istruzioni, tuttavia, sono lungi da essere intuitive. In primo luogo, un'espressione può essere composta di una o più espressioni secondarie separate da virgola. Consideriamo, ad esempio, la seguente espressione:

```
if (++x, --y, cin.good())
```

L'istruzione condizionale *if* contiene tre espressioni separate da virgola. Il compilatore C++ si assicura che ciascuna delle espressioni venga valutata e che i relativi effetti avvengano. In questo caso, la variabile *x* viene incrementata e la variabile *y* è decrementata. Tuttavia, il valore dell'intera espressione separata da virgola è soltanto il risultato dell'espressione più a destra. Di conseguenza, l'istruzione if di sopra è vera solo se il metodo *cin.good()* ritorna *true*. Consideriamo invece la seguente espressione:

```
int j=10;
int i=0;
while( ++i, --j)
{ //... }
```

Nel ciclo *while*, la variabile *i* viene incrementata mentre la variabile *j* è decrementata. Il ciclo prosegue fintanto che il valore di *j* è maggiore di *0*.

Come richiamare una funzione prima dell'avvio di un programma?

Alcune applicazioni richiedono di invocare funzioni prima che la parte principale del programma venga eseguita. Ad esempio, la funzione di log deve essere richiamata prima dell'esecuzione dell'applicazione. Il metodo migliore per invocare queste funzioni è di collocare la loro invocazione all'interno del costruttore di un oggetto globale dell'applicazione, questo perché gli oggetti globali sono costruiti prima che l'esecuzione passi al programma. Pertanto, le funzioni invocate in un costruttore di un oggetto globale verranno invocate prima della funzione *main()*. Ad esempio:

```
class Login
{
public:
Login()
{
attiva_login();
}
};
Login log; // istanza globale
int main()
{ record * prec=Leggi_login();
  //.. codice dell'applicazione
}
```

L'oggetto globale log viene costruito prima che l'esecuzione passi alla funzione main(). Durante la costruzione, l'oggetto invoca la funzione *attiva_login()*. Quando l'esecuzione passa alla funzione main() dell'applicazione, è possibile quindi leggere i dati dal log file.

Cosa sono i puntatori ai membri?

Una classe può avere due categorie di membri: *membri funzioni* (metodi) e *membri dati* (variabili membro). Inoltre, esistono due tipi di puntatori: puntatori a metodi e puntatori a variabili membro. Gli ultimi sono i più diffusi poiché, generalmente, le classi non hanno variabili membro pubbliche. Tuttavia, se utilizziamo codice C esistente che contiene dichiarazioni struct o classi che hanno variabili membro dati pubbliche, i puntatori a queste variabili possono rivelarsi molto utili. La sintassi per i puntatori a membri è una delle più complicate del linguaggio C++. L'uso di tali puntatori, però, rappresenta una caratteristica molto potente del linguaggio, perché permette di invocare una funzione membro di un oggetto senza conoscerne il nome. Analogamente, è possibile usare un puntatore per esaminare o modificare il valore di una variabile membro senza conoscerne il nome.

L'esperto risponde...

FLASH e VB

Gentile Giuliano Uboldi, sono un programmatore "in erba" di 14 anni ed avrei bisogno del tuo aiuto: nel numero 10 di ioProgrammo (12 /02), ho trovato molto interessante l'articolo sull'integrazione di Flash con VB e provando a realizzare, per cominciare, il programmino sono incappato in un errore: innanzi tutto premetto che io utilizzo VB6 e Flash MX su una piattaforma WIN 2000. Ora, dopo aver compilato la parte Flash e quella VB ho provato a far partire il programma dal VB stesso, ma il debugger trova un errore nella *Sub Aggiorna* alla stringa:

```
"Text1.Text = Flash1.GetVariable("areaTesto.text"
)" (Text1 è il Testo dell'articolo);
```

l'errore riscontrato è:

```
"Errore di runtime '-2147467259(80004005)
Metodo 'GetVariable' dell'oggetto
'IShockVaweFlash' non riuscito
```

La ringrazio e la prego di inviarmi o pubblicare al più presto una sua risposta. Distinti saluti.

Alberto Santini

Risponde: Giuliano Uboldi

Caro Alberto, devo ammettere che, prelevando direttamente i file dell'esempio dal cd ed utilizzandoli in Win 2000 si ottiene l'errore di runtime che hai riportato. Ciò è dovuto ad un errato riferimento di VisualBasic rispetto al componente da utilizzare per eseguire l'applicazione in oggetto. Per correggere questo comportamento si deve, dal menù *Progetto->Riferimenti*, controllare che il riferimento al controllo Shockwave Flash sia legato ad un file *.ocx* piuttosto che ad uno *.oca*. Se infatti il riferimento al componente *ShockWave* Flash è legato ad un .oca, anche se con lo stesso nome, il programma non funziona. Tutto ciò che devi fare quindi è selezionare dall'elenco che appare, selezionando le voci di menù sopra indicate, l'altro componente ShockWave *Flash.ocx* che sicuramente troverai. Vedrai che con questa semplice correzione riuscirai a far girare il tutto. Un caro saluto.

MFC e WIN32

Un saluto e molti complimenti per la rivista che leggo bramosamente ogni mese. Domandina: che differenza c'è fra MFC e WIN32 (visualC++)? E' una questione di librerie, o anche di architettura?

Alessandro Danilo Brevi

Per realizzare un'applicazione che giri su piattaforma Windows, il compilatore deve interfacciarsi con il sistema operativo attraverso delle API *(Application Programming Interface)* che nel caso di Windows prendono proprio il nome di Win32 (dove 32 sta per 32bit). Inizialmente, le API di Windows vennero sviluppate come un insieme di procedure separate e rilasciate attraverso una piattaforma chiamata *Windows API SDK (Software development Kit)*, inizialmente con Api a 16 bit. La realizzazione delle applicazioni sotto Windows era pertanto realizzata mediante l'uso di queste "procedure" di sistema. Un esempio di applicazione realizzata sono le prime versioni di Word per Windows. Ad un certo punto, visto il diffondersi della programmazione object oriented, un team venne incaricato di sviluppare un framework ad oggetti in cui venissero incapsulate le API di sistema. Questo per facilitare la realizzazione delle applicazioni e per permettere un elevato riutilizzo del codice. Il framework in questione è appunto MFC *(Microsoft Founfation Classes)*. Visto il successo di questo framework, ad un certo punto si decise di riscrivere le API di Windows in chiave MFC. Pertanto, oltre a costituire un potente framework per lo sviluppo di applicazioni in ambiente Windows, le MFC rappresentano oggi le API stesse del sistema, e cioè il modo privilegiato di interfacciarsi con esso attraverso un'architettura ad oggetti.

Applicazioni prive di form

Cara redazione, sono abbonato a ioProgrammo ormai da tre anni, e penso che questo segno di stima mi possa esimere dal formularvi i complimenti di rito. Arrivo subito al dunque: vorrei realizzare un'applicazione in Visual Basic, che non presenti alcuna form. Vorrei cioè che l'applicazione avesse le sembianze di un semplice message box... so che può sembrare una richiesta un po' strana, ma avrei bisogno di una risposta in tal senso. Vi ringrazio e vi auguro buon lavoro.

Stefano

Per ottenere il risultato che desideri, apriamo un nuovo progetto Visual Basic e andiamo ad eliminare tutte le form incluse nel progetto (essendo il progetto nuovo, si tratterà di eliminare semplicemente *form1*). Dal menu *Progetto*, selezioniamo la voce *Proprietà* e, dal combobox *"Oggetto di avvio"*, selezioniamo *"Sub Main"*. A questo punto, aggiungiamo un nuovo modulo standard al progetto e, all'interno di questo, creiamo una procedura chiamata *"Main"* attraverso un codice simile a quello che puoi trovare di seguito

```
Private Sub Main()
    MsgBox("Ciao!")
End Sub
```

Ovviamente, al posto del Message Box, puoi aggiungere qualsiasi codice, ed il tutto sarà eseguito senza che venga visualizzata alcuna form.

Per contattarci:
e-mail: iopinbox@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano

Gamasutra

The Art & Science of Making Games http://www.gamasutra.com/

Chiacchierando con amici, colleghi e conoscenti interessati al mondo della programmazione, ho avuto modo di verificare come le mie radici di informatico affondino in un terreno fertile e condiviso. Correva la seconda metà degli anni '80, ed io ero ancora un bambino sotto i dieci anni. Già qualche volta, entrando di soppiatto nelle proibitissime sale giochi, ero rimasto affascinato da quegli strani "scatoloni con la TV" che monopolizzavano l'attenzione di molti ragazzi un po' più grandi di me. Ricordo ancora, con grande nostalgia, i diabolici piani che organizzavo insieme agli amici d'infanzia per andare a giocare liberamente senza essere scoperti dai nostri genitori. Evidentemente, però, non siamo stati molto abili, giacché in famiglia qualcuno intese immediatamente la mia passione per i giochi elettronici. Per la prima comunione, infatti, ricevetti in regalo uno storico Atari 2600, che non avevo richiesto e di cui nemmeno sospettavo l'esistenza. Quel regalo, visto col senno di poi, mi ha cambiato la vita. Di lì in poi, la mia passione per i videogiochi è cresciuta in maniera esponenziale. Prima il Commodore 64, poi numerose console quali il Master System, il Mega Drive ed il Game Gear (sì, ero un partigiano della fazione Sega, ai tempi della lotta contro Nintendo), quindi l'Amiga ed infine il PC Intel. Già ai tempi del Commodore 64 mi domandavo quali fossero gli "ingredienti" di un videogioco. Chi crea i videogiochi? Come è possibile costruire qualcosa di simile? Nessuno dei miei compagni di giochi sapeva dare risposta a queste domande. Così come le popolazioni primitive, nella loro ignoranza, attribuirono il fulmine al potere degli dei, per noi il "creatore di videogiochi" era una figura mitologica ed inarrivabile. Dire "da grande voglio fare il creatore di videogiochi" era proprio come voler divenire astronauta, calciatore, pilota di Formula 1 o Presidente della Repubblica. Passavamo le serate estive a sognare il nostro videogioco ideale, come se di lì a poco l'avremmo realizzato. A dieci anni circa, sapevo qualcosa di programmazione applicata al Commodore 64, ma nulla di esattamente cosciente. Che un videogioco potesse essere realizzato in casa, lo scoprii poco tempo dopo. Il merito lo devo dare proprio ad una rivista e alla sua rubrica *"Talent Scout"*, che recensiva le attrazioni videoludiche sviluppate dai lettori. Scoperta la possibilità ed esplorati i mezzi, non so quantificare il tempo speso sulla tastiera, a digitare istruzioni Basic e Pascal. Non credo di aver mai abbandonato il sogno di sviluppare videogiochi, benché la passione per la programmazione videoludica sia poi mutata lentamente in amore per l'informatica, in termini più generali. Chissà quanti di quei bambini sognatori, oggi, sono divenuti informatici, programmatori, ingegneri, amministratori di sistema e altro ancora. Sì, le scappatelle in sala giochi e quello strano Atari 2600 mi hanno cambiato la vita. Nonostante l'età avanzi senza sosta, il sogno nel cassetto non è mai scomparso. Sviluppare videogiochi, forse, non sarà mai il lavoro che mi darà il pane, ma sono certo che rimarrà sempre uno degli hobby prediletti. Oggi, ad ogni modo, realizzare un videogioco è tutt'altro che semplice. Il mito del programmatore solitario è rilegato al passato. Lo sviluppo di un videogioco, con le possibilità dei tempi nostri, richiede personale e mezzi che si avvicinano più al settore cinematografico che non a quello informatico. Nessuno, ad ogni modo, ci vieta di portare avanti i nostri amati progetti. Al mondo sono ancora tanti quelli che bramano il parto del videogioco dei loro sogni. Che lo si faccia per professione o per hobby, Gamasutra rimane un riferimento imperdibile. "The Art & Science of Making Games", ossia "l'Arte e la Scienza di Fare Giochi", recita la striscia sopra il logo del sito. Ho letto pochi motti così ispirati. Progettare e realizzare videogiochi è qualcosa a metà strada tra l'arte e la scienza. Gamasutra è una sterminata raccolta di articoli che coprono tutte le fasi dello sviluppo di un videogioco moderno, dal design alla programmazione, dalla grafica all'audio, fino a giungere alle questioni legali sollevate dalla pubblicazione e dalla diffusione di un gioco. Nessuna piattaforma videoludica è estranea a Gamasutra. Troverete informazioni preziose per sviluppare giochi destinati, indifferentemente, ai personal computer, alle console e ai dispositivi portatili. Più che altro, la sezione rivolta agli autori di codice si concentra principalmente su tecniche algoritmiche di applicazione universale. Non mancano articoli di riflessione sul mondo videoludico, news sempre aggiornate, interviste con personalità del settore, calendari di eventi, bacheche per gli annunci ed altro ancora. In poche parole, Gamasutra è una fresca rivista elettronica che apre un'ampia finestra sull'affascinante universo del game programming. Una paio di note negative: per accedere al materiale disponibile nel sito è sempre necessario registrarsi, ed il layout delle pagine non è molto commestibile. Mali perdonabili, davanti alla bontà e alla ricchezza dei contenuti. Bookmark d'obbligo per ogni game programmer o aspirante tale.

Carlo Pelliccia